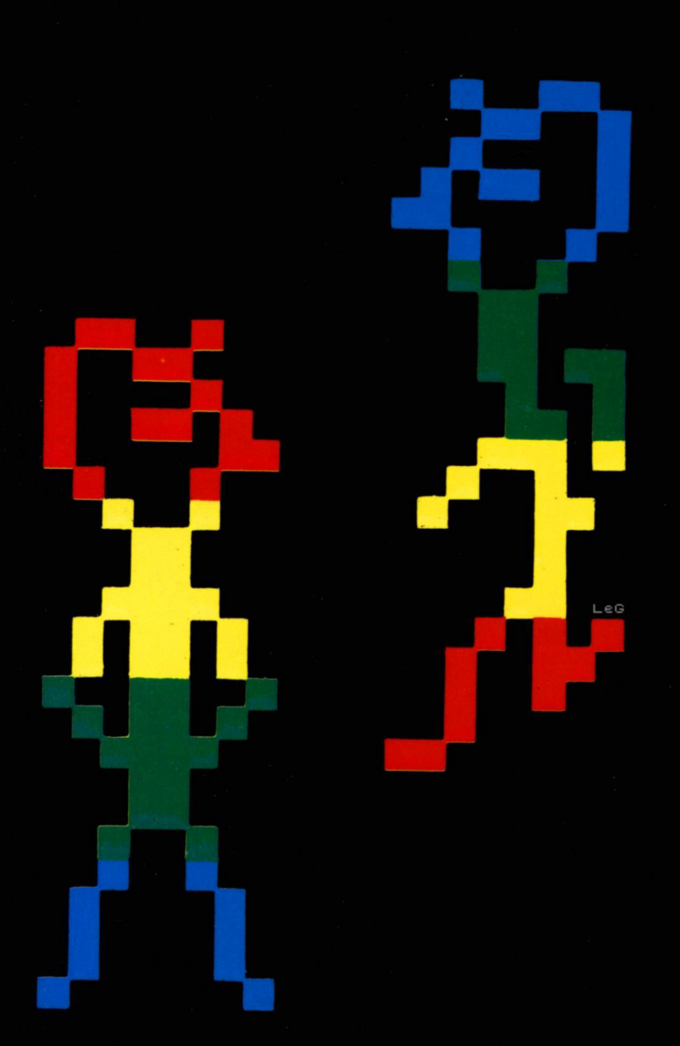
Cómo usar los colores y los gráficos en el Spectrum

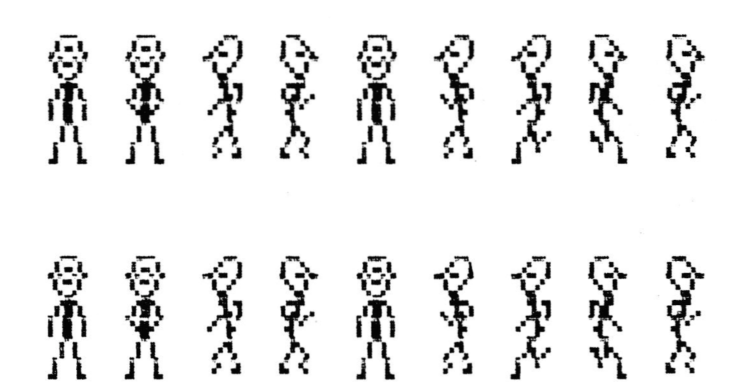


- Alta resolución
- Peek y Poke
- Movimiento
- Casualidad
- Gráficos
- Color

ANTONIO BELLIDO



Cómo usar **los colores y los gráficos** en el **Spectrum**



ANTONIO BELLIDO

Cómo usar **los colores y los gráficos** en el **Spectrum**

TERCERA EDICION

1984



MADRID

IMPRESO EN ESPAÑA
PRINTED IN SPAIN

ISBN: 84-283-1311-3

Depósito Legal: M-31365-1984



Magallanes, 25 - 28015 - MADRID (5-3385)

ALCO, artes gráficas. Jaspe, 34 - 28026 - MADRID

Este libro está escrito como una introducción al uso de los gráficos y el color en los computadores populares y ha sido desarrollado en base a un ZX Spectrum. Su objetivo es ayudar al lector a transformar sus ideas en programas llenos de color y movimiento.

TODOS LOS PROGRAMAS QUE SE DESARROLLAN EN ESTE LIBRO SE INCLUYEN EN UN CASETE QUE PUEDE ADQUIRIRSE OPCIONALMENTE. SOLICITELO A SU PROVEEDOR HABITUAL O A PARANINFO, S.A.

A través de "La pequeña gran puerta", primer libro de esta serie, me propuse mostrar en qué forma "razona" un computador, partiendo de los supuestos de no conocer nada sobre el lenguaje de programación BASIC, ni del idioma inglés.

Un segundo libro —"Cómo programar su Spectrum"— está dedicado a enseñar, sin dificultad, desde cómo manejar el teclado hasta la programación en BASIC.

"Cómo usar los colores y los gráficos del Spectrum" está diseñado como complemento del libro anterior y su objetivo es ayudar al lector en el camino de transformar sus ideas en programas plenos de color y movimiento.

Aparentemente los programas que se desarrollan en este libro son simples juegos. No los considere de esta forma ya que, según mi experiencia, los programas que podrían tener el calificativo de "técnicos", suelen ser más fáciles de concebir debido a que, normalmente, están sometidos a fórmulas y procesos muy determinados, en tanto que los programas de "juegos" son trozos de su propia imaginación.

Los programas aquí presentados son maquetas cuya misión es abrir el campo de posibilidades que Vd. tiene delante, por tanto le sugiero que cambie y pruebe toda cosa que crea que mejorará los resultados.

Por último, le indico que este libro está escrito basándose en un Spectrum de 16K, lo cual significa que todo el contenido es igualmente útil para cualquier tipo de Spectrum.

Si consideramos la pantalla como una pizarra donde el Spectrum nos va a dar su respuestas a nuestras cuestiones, vamos a empezar por conocer en que forma la máquina usa esa "pizarra".

Cuando se conecta el computador al televisor, el "cerebro" de aquél divide de forma inmediata, y sin que se pueda apreciar, la pantalla de éste en una malla de 45.086 cuadritos o, lo que es igual, 256 cuadritos horizontales por 176 cuadritos verticales (ver figura 1 en un desplegable entre las páginas 16 y 17). Estos cuadritos pueden estar "encendidos" o "apagados". Y según la combinación de cuadritos encendidos y apagados se verá en la pantalla una figura u otra.

Siguiendo el símil de la pizarra, la instrucción INK equivale al color de la tiza con la que vamos a escribir y la instrucción PAPER nos indica el color de fondo de la pizarra.

Evidentemente, el término **apagado** significa que el cuadrito en cuestión mantiene el color dado por PAPER, mientras que **encendido** significa que toma el color dado por INK.

Para aclarar este concepto tomaremos, por ejemplo, las instrucciones INK 1 y PAPER 2. Debido a la primera los cuadritos encendidos tomarán el color azul, los apagados mantendrán el color general del fondo que, en este caso, es rojo y que es impuesto por la segunda.

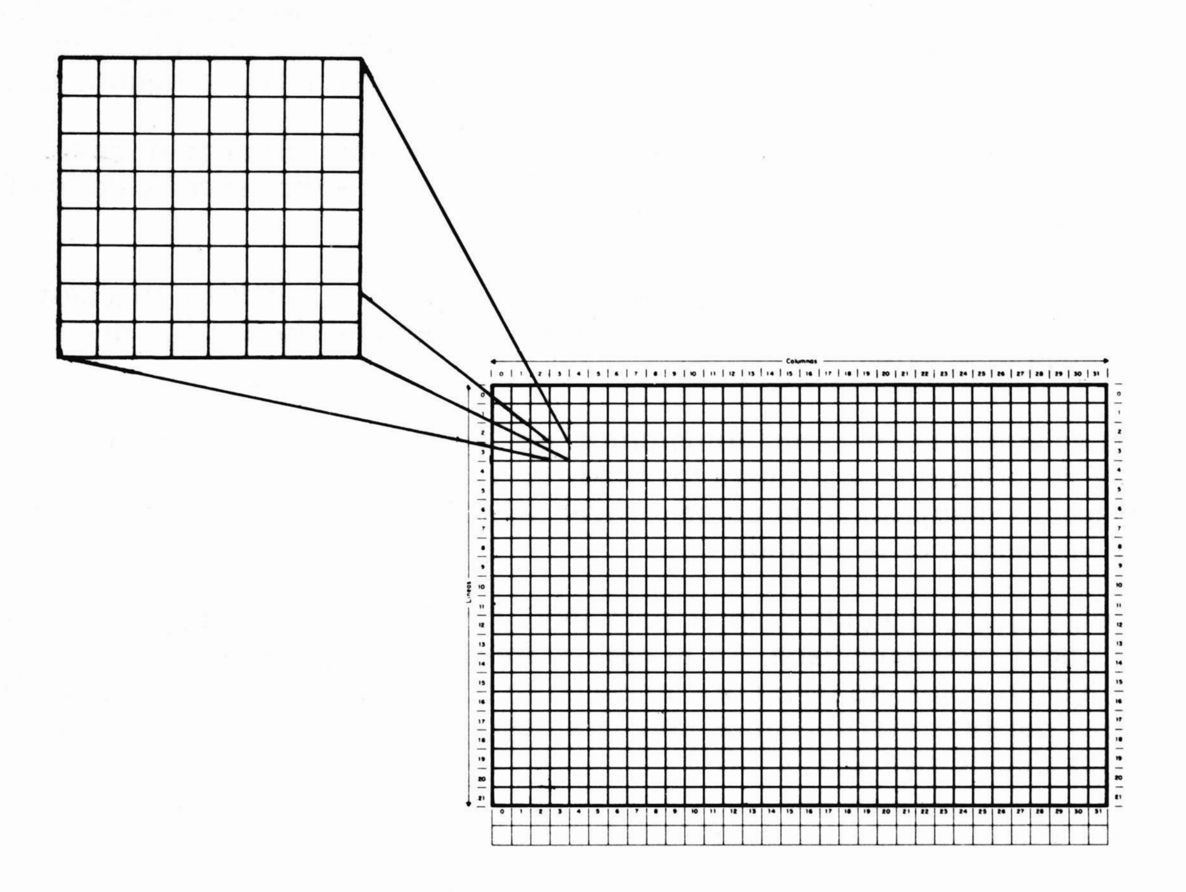
Es claro que utilizando colores para INK y PAPER con poco contraste, apenas se podrá distinguir nada. En el caso extremo en el que INK y PAPER tengan el mismo color, no habrá diferencia entre cuadritos encendidos y apagados. En otras palabras escribiríamos con tiza negra sobre una pizarra negra, si hubieramos elegido INK Ø, PAPER Ø.

Hasta ahora hemos visto cómo "pinta" el Spectrum sobre la pantalla, veamos ahora cómo escribe.

Anteriormente estudiamos que el Spectrum "contempla" la pantalla como un conjunto de 45.086 cuadritos (256 x 176), pues bien, el Spectrum tiene guardados en su memoria un conjunto de caracteres, que le es propio por diseño, tales como todo el abecedario en mayúsculas y minúsculas, los signos de puntuación, números, etc. y además los tiene definidos en pequeñas mallas de 8 cuadritos horizontales por 8 verticales, de tal forma que, si tuviera que mos-

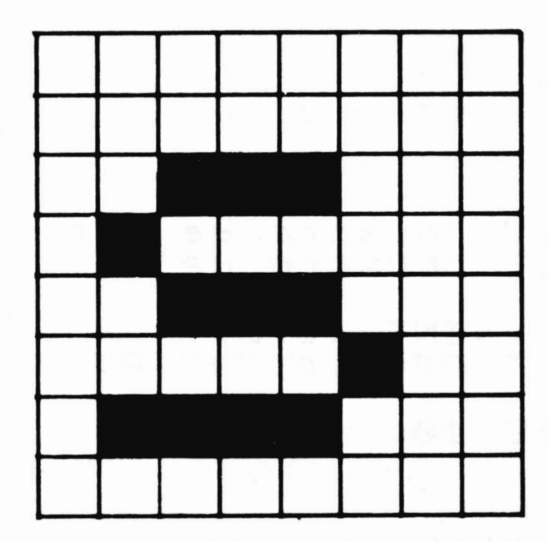
trar en la pantalla uno cualquiera de estos caracteres predefinidos tomaría, dentro de los 45.086 cuadritos totales, una zona de 8 x 8 cuadritos dentro de la cual encendería unos y apagaría otros, mostrando, de esta forma, la figura del carácter deseado.

REPRESENTACION GRAFICA DE LA PANTALLA



De lo dicho hasta aquí se desprende que la pantalla gobernada por el Spectrum nos puede dar 32 caracteres por línea (256/8) y un total de 22 líneas (176/8).

Como ya sabemos, con la instrucción PRINT ordenamos a la máquina que imprima en pantalla números y caracteres en general. De acuerdo con lo visto anteriormente, cuando usamos PRINT para escribir un carácter en la pantalla, lo que estamos haciendo en realidad es decirle al computador que nos muestre cual es el conjunto de cuadritos encendidos y apagados que corresponde a ese caracter, y que él tiene "guardado" en su memoria. Veamos cuál es la combinación correspondiente a la **s** por ejemplo:



Estos son los caracteres que están predefinidos en la máquina por el diseñador.

El usuario puede definir sus propios caracteres o modificar los predefinidos. Esto lo veremos más tarde.

De ahora en adelante, cuando hagamos referencia a caracteres en general estaremos hablando del espacio ocupado en pantalla por mallas de 8 x 8 cuadritos. Un espacio en blanco es una malla de 8 x 8 con sus 64 cuadritos apagados.

Recordamos que las filas van aumentando de 0 a 21 de arriba a abajo y las columnas de 0 a 31 de izquierda a derecha.

Para cerrar este capítulo vamos a realizar un ejercicio de impresión en pantalla, interesante desde el punto de vista de la estética y, en muchos casos, de gran utilidad.

El siguiente programa nos permite imprimir asteriscos (*) en cualquier punto de la pantalla:

```
10 INPUT "numero de linea?";l
20 INPUT "numero de columna?";
c
30 PRINT AT l,c;"*"
40 GO TO 10
```

1

Para imprimir un asterisco en cualquier punto de la mitad izquierda de la pantalla y su simétrico en la mitad derecha, hay que considerar que, si las coordenadas de entrada son **l** y **c**, para su simétrico serán **l** y **31-c**:

```
10 INPUT "numero de linea?";l
20 INPUT "numero de columna?";
c
25 LET c=INT (c/2)
30 PRINT AT l,c;"*";AT l,31-c;
"*"
40 GO TO 10
```

2

Con la línea 25 limitamos **c** a la mitad izquierda de la pantalla.

Para conseguir que el asterisco se imprima según cuatro planos de simetría:

```
10 INPUT "numero de linea?";l
20 INPUT "numero de columna?";
c
25 LET l=INT (l/2): LET c=INT
(c/2)
30 PRINT AT l,c;"*";AT l,31-c;
"*";AT 21-l,c;"*";AT 21-l,31-c;"
*"
40 GO TO 10
```

3

Con la línea 25 limitamos **l** y **c** al cuadrante superior izquierdo.

Supongamos que deseamos ponerle un marco a la pantalla, para ello escribiríamos un programa similar a éste:

```
   10 PRINT "********************
************"
   20 FOR i=1 TO 20
   30 PRINT "*";TAB (31);"*"
   40 NEXT i
   50 PRINT "********************
************"
```

4

Este marco hecho con asteriscos no queda mal, pero el Spectrum nos provee de 16 caracteres gráficos predefinidos, es decir, están memorizados dentro de la máquina. Ocho de estos caracteres están a la vista en la primera fila de teclas de la máquina y los restantes son, simplemente, sus inversos.

Para obtener los ocho primeros basta con pasar al modo gráfico **(cursor G)** y apretar, a continuación, la tecla correspondiente al caracter deseado. Así, apretaremos simultáneamente las teclas **CAPS SHIFT** y **GRAPHICS** para cambiar el cursor a G y, en estas condiciones, al apretar cualquier tecla con caracter gráfico, nos producirá en la pantalla la impresión de ese carácter.

Para conseguir las inversas tendremos que pasar el cursor igualmente a G y, una vez hecho esto, apretar simultáneamente CAPS SHIFT y la tecla correspondiente.

Compare los gráficos que aparecen en pantalla con los siguientes ejemplos:

Pulse CAPS SHIFT y GRAPHICS. (Ahora el cursor está en G). Pulse la tecla con el número 5, la tecla con el número 6, la tecla con el número 2 y la tecla con el número 8.

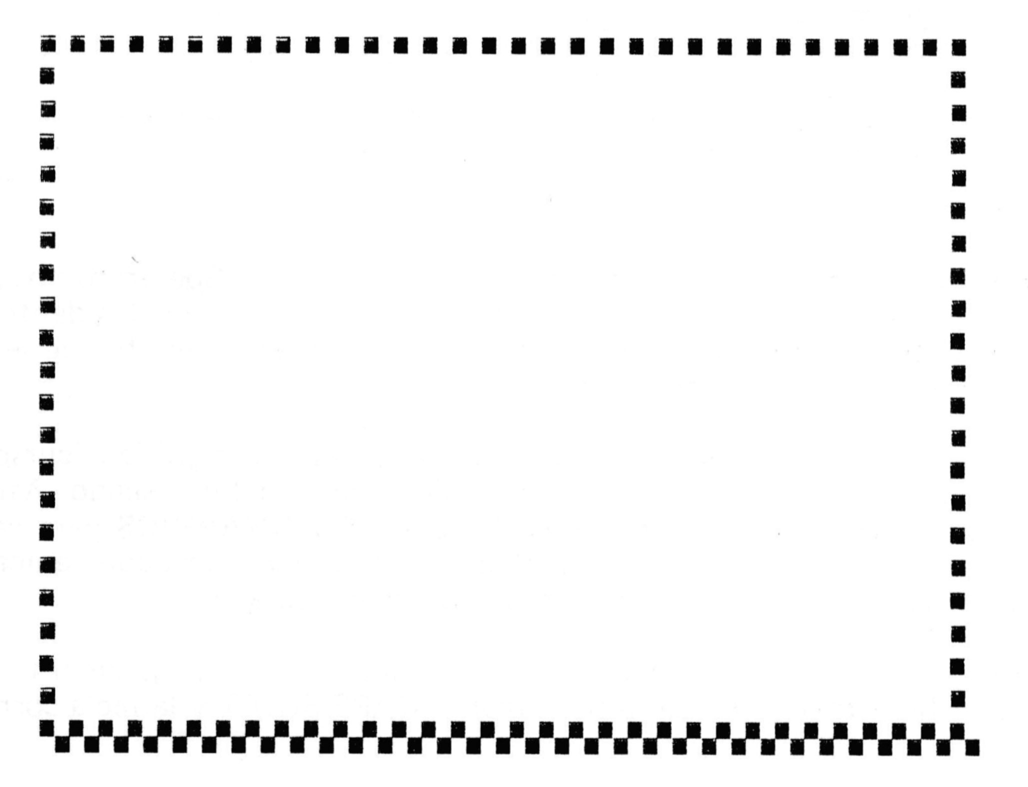
A continuación pulse 2 ó 3 veces SPACE y pulse nuevamente las mismas teclas 5, 6, 2 y 8, pero apretando simultáneamente CAPS SHIFT.

Como comprobará, los gráficos que aparecen en pantalla son exactamente los inversos a los que pulsó anteriormente.

En el programa del ejemplo anterior, podríamos hacer un marco más "bonito" con este desarrollo:

```
10 FOR i=0 TO 31
20 PRINT "▘ ";
30 NEXT i
40 FOR i=1 TO 20
50 PRINT "▘ ";TAB (31);"▘ "
60 NEXT i
70 FOR i=0 TO 31
80 PRINT "▘▗";
90 NEXT i
```

5



En otro orden de cosas es necesario saber cómo transformar números en caracteres. Cuando nosotros decimos que la tercera letra del alfabeto es la C, hemos aprovechado la relación existente entre la posición que ocupa en el alfabeto y la propia letra. Pues bien, algo similar ocurre con el conjunto de caracteres que maneja el Spectrum, gracias a la función CHR$.

Si nosotros tecleamos PRINT CHR$ (67) obtendremos el caracter que el Spectrum memoriza en la posición 67 dentro de su conjunto de caracteres (página 183 del manual).

Para ver en pantalla este conjunto corramos este programa.

```
10 FOR i=32 TO 255
20 PRINT CHR$ (i);
30 NEXT i
```

6

Con lo cual obtendremos:

```
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]↑_
£abcdefghijklmnopqrstuvwxyz{|}~©
 ▝▘▀▗▐▚▜▖▞▌▛▄▟▙█ABCDEFGHIJKLMNOP
QRSTURNDINKEY$PIFN POINT SCREEN$
 ATTR AT TAB VAL$ CODE VAL LEN S
IN COS TAN ASN ACS ATN LN EXP IN
T SQR SGN ABS PEEK IN USR STR$ C
HR$ NOT BIN OR AND <=>=<> LINE T
HEN TO STEP DEF FN CAT FORMAT MO
VE ERASE OPEN # CLOSE # MERGE VE
RIFY BEEP CIRCLE INK PAPER FLASH
 BRIGHT INVERSE OVER OUT LPRINT
LLIST STOP READ DATA RESTORE NEW
 BORDER CONTINUE DIM REM FOR GO
TO GO SUB INPUT LOAD LIST LET PR
USE NEXT POKE PRINT PLOT RUN SAV
E RANDOMIZE IF CLS DRAW CLEAR RE
TURN COPY
```

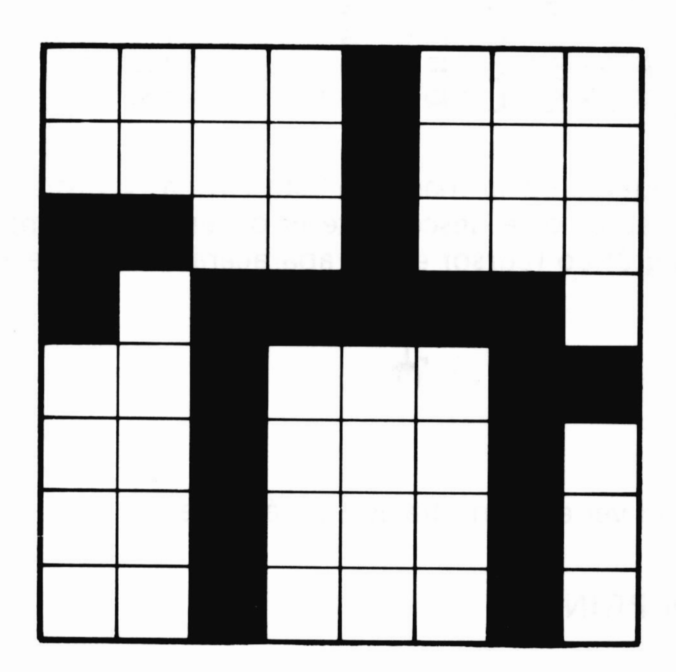
En realidad los 32 primeros códigos y los que siguen al 143 no son propiamente caracteres predefinidos y específicamente los 91 últimos corresponden a palabras sentencias del BASIC, que el Spectrum considera caracteres sin serlos.

Hasta aquí lo que el Spectrum nos ofrece en su memoria por estar así diseñado, a partir de ahora veremos lo que nosotros podemos lograr de él.

Supongamos que queremos crear un caracter gráfico que represente un jinete a caballo de tal forma que, siempre que apretemos la tecla adecuada y en modo gráfico, nuestro jinete aparezca.

En primer lugar no debemos olvidar que los caracteres —cualquier carácter, predefinido o no— deben estar comprendidos en una malla de 8 cuadritos horizontales por 8 verticales y que el carácter propiamente dicho, se logra con una adecuada combinación de cuadritos encendidos (color INK) y apagados (color PAPER).

Si representamos caballo y jinete en el cuadro correspondiente, tendremos:



Los cuadritos sombreados corresponden a los que hemos dado en llamar "encendidos", y que en lo que sigue los representaremos por 1 y los restantes son los apagados y que designamos por 0. Esto nos dará la siguiente combinación de 0 y 1:

Línea 0	0 0 0 0 1 0 0 0
Línea 1	0 0 0 0 1 0 0 0
Línea 2	1 1 0 0 1 0 0 0
Línea 3	1 0 1 1 1 1 1 0
Línea 4	0 0 1 0 0 0 1 1
Línea 5	0 0 1 0 0 0 1 0
Línea 6	0 0 1 0 0 0 1 0
Línea 7	0 0 1 0 0 0 1 0

La forma de crear este carácter, parte de cualquier letra desde la A a la U y por la sucesiva modificación de cada línea de 8 cuadritos horizontales de la malla de 8 x 8 donde esta inscrito el carácter seleccionado. Por tanto, hay que elegir una letra, la A p.e., y comenzar modificando la línea 0 de la malla de 8 x 8, después la 1 y así hasta la 7.

Estas modificaciones se realizan con POKE USR y la última combinación de 0 y 1 en que acabamos de convertir el jinete y su caballo, de la siguiente manera:

```
10 POKE USR "a"+0,BIN 00001000
20 POKE USR "a"+1,BIN 00001000
30 POKE USR "a"+2,BIN 11001000
40 POKE USR "a"+3,BIN 10111110
50 POKE USR "a"+4,BIN 00111111
60 POKE USR "a"+5,BIN 00101010
70 POKE USR "a"+6,BIN 00100010
80 POKE USR "a"+7,BIN 00100010
```

[7]

Una vez introducida en la máquina esta rutina, y hasta que no se cambie la definición del carácter o se desconecte el ordenador, siempre que se apriete la tecla A en modo gráfico (cursor en G) aparacerá este jinete a caballo:



El proceso para ver en pantalla este carácter es:

1°.– Apretar PRINT.

2°.– Pasar el cursor a modo gráfico (CAPS SHIFT y GRAPHICS).

3°.– Apretar la tecla A.

En estas condiciones –con la rutina de nuestro jinete en la tecla A– corra el siguiente programa:

```
10 FOR i=0 TO 31
20 PRINT AT 0,31-i;" ";
30 FOR j=0 TO 15: NEXT j
40 NEXT i
```

[8]

Supongamos que nos disponemos a escribir una carta. Lo primero que haremos, lógicamente, será proveernos de papel y lápiz.

Algo similar sucede con el Spectrum.

Con las instrucciones **INK** (TINTA) y **PAPER** (PAPEL) definimos el color de la tinta y el papel sobre el que vamos a escribir y para todo lo que siga a esos comandos, pero no a lo ya existente en la pantalla.

Ejemplo:

```
  10 INPUT "introduzca el numero
 del color del papel";p: PAPER p
  20 CLS
  30 INPUT "introduzca el numero
 del color de la tinta";t: INK t
  40 CLS
  50 PRINT "Los colores del Spec
trum"
  60 GO TO 10
```

9

En cambio, si INK y PAPER están contenidos en una instrucción que se inicie con la sentencia PRINT, estos colores son mantenidos hasta el momento en que la instrucción ha sido ejecutada, volviendo INK y PAPER a los colores que tuvieran anteriormente.

Compare el siguiente ejemplo con el anterior:

```
  10 INPUT "introduzca el numero
 del color del papel";p
  20 CLS
  30 INPUT "introduzca el numero
 del color de la tinta";t
  40 CLS
  50 PRINT INK t; PAPER p;"Los c
olores"; INK p; PAPER t;" del Sp
ectrum"
  60 GO TO 10
```

10

Observe como se comporta el color de fondo de la pantalla en ambos casos.

Con el comando CLS se "apagan" todos los cuadritos de la pantalla, con lo cual todo se torna en el color de PAPER, dando como resultado un borrado de la pantalla.

Con BORDER coloreamos toda la zona que circunvala el área de trabajo del Spectrum. En otras palabras, con BORDER se da color a todo lo que está fuera de los 45.086 cuadritos que controla el ordenador.

Ejemplo:

```
 10 INPUT "numero del color del
borde?";b: CLS
 30 BORDER b
 40 GO TO 10
```

11

Observe como el color de BORDER afecta también a las dos líneas últimas de la pantalla, para ello pruebe un BORDER negro y así forzará al ordenador a imprimir sus mensajes con tinta blanca.

Con INVERSE 1 conseguimos, para aquellos caracteres que sigan a esta instrucción, que todos los cuadritos cuya situación normal con respecto a esos caracteres debían estar encendidos, se apaguen y los que debían estar apagados se enciendan.

Con INVERSE 0 volvemos a la combinación de cuadritos apagados y encendidos estándar, para los caracteres afectados.

Ejemplo:

```
  10 INPUT "situacion de INVERSE
?";i: INVERSE i
  20 PRINT "Los colores del Spec
trum"
  50 GO TO 10
```

12

Antes de estudiar la sentencia OVER 1, recordemos que al imprimir un carácter en una posición de pantalla donde hay otro carácter, este último desaparece para dar lugar al nuevo.

Ejemplo:

```
10 PRINT AT 10,15;"/"
20 PRINT AT 10,15;"\"
```

13

Al correr este programa finalmente nos quedará en la pantalla el símbolo \.

Si añadimos:

5 CLS
7 OVER 1

nos quedará:

14

```
 5 CLS
 7 OVER 1
10 PRINT AT 10,15;"/"
20 PRINT AT 10,15;"\"
```

Corriendo este nuevo programa, resultará que la impresión en pantalla estará compuesta por dos barras que se cruzan y con el punto de corte de ambas en color PAPER.

Esta sentencia tal vez parezca complicada pero en definitiva trabaja de la siguiente manera: si dos caracteres coinciden en una misma posición de pantalla, la sentencia OVER 1 haría que todos los cuadritos apagados se mantengan apagados, mientras que los cuadritos encendidos del último carácter continúen encedidos si coinciden con cuadritos apagados del primer carácter, o se apaguen si coinciden con cuadritos encendidos.

Para anular la condición de sobreimpresión basta con la instrucción OVER 0.

La forma de conseguir un efecto de parpadeo partiendo de las características de sobreimpresión de OVER ya estudiadas, se basa en una rutina de este tipo:

15

```
  10 PRINT AT 10,3;"Los colores
del Spectrum"
  20 OVER 1
  30 FOR x=0 TO 100: NEXT x
  40 GO TO 10
```

Con BRIGHT 1 todos los cuadritos encendidos de los caracteres que sigan a esta instrucción aparecerán sobreiluminados, hasta que un BRIGHT 0 cancele esta situación.

Con FLASH 1 todos los cuadritos encendidos de los caracteres que sigan a esta instrucción, se apagarán y encenderán intermitentemente, produciendo un efecto de parpadeo hasta que un FLASH 0 cancele esta situación.

Estos dos últimos comandos pueden trabajar juntos para obtener una situación de parpadeo y sobreiluminación.

Ejemplo:

```
   5 INK 1: PAPER 7
  10 PRINT AT 0,3;"Los colores d
el Spectrum"
  12 PRINT AT 1,3;"NORMAL"
  15 PAUSE 100
  20 BRIGHT 1
  25 PRINT AT 5,3;"Los colores d
el Spectrum"
  27 PRINT AT 6,3;"SOBREBRILLO"
  30 PAUSE 100
  35 BRIGHT 0: FLASH 1
  40 PRINT AT 10,3;"Los colores
del Spectrum"
  42 PRINT AT 11,3;"PARPADEO"
  45 PAUSE 100
  50 BRIGHT 1
  55 PRINT AT 15,3;"Los colores
del Spectrum"
  60 PRINT AT 16,3;"SOBREBRILLO
Y PARPADEO"
  90 FLASH 0: BRIGHT 0
```

16

De la misma forma en que INK y PAPER trabajan dentro de una instrucción PRINT dando color sólo a los caracteres de esa instrucción, también responderán INVERSE, OVER, FLASH y BRIGHT.

Ejemplo:

```
  10 PRINT "Los colores"
  20 PRINT BRIGHT 1;AT 0,12;"del
Spectrum"
```

17

Un resumen de lo aquí tratado y unas consideraciones de carácter general darán por finalizada esta primera parte.

El Spectrum maneja la pantalla del televisor considerando la parte central como una malla de 256 cuadritos horizontales por 176 verticales. El color de esta zona se controla con la instrucción PAPER. El color del resto de la pantalla, que rodea a la anterior, se controla con BORDER.

Inicialmente los colores de estas dos áreas son blancos. Los caracteres se imprimen en la zona central de la pantalla y sobre el color fijado por el comando PAPER. La figura de los caracteres se obtienen sobre mallas de 8 x 8 cuadritos, lo cual da lugar a un máximo de 22 líneas de 32 caracteres cada una, es decir 704 (22 x 32) posiciones posibles de carácter.

Hay dos líneas inferiores que quedan reservadas para la entrada de datos y la emisión de mensajes por parte de la máquina.

Cada malla de carácter de 8 x 8 cuadritos tiene unas características propias que son conocidas como sus **atributos.** Estos atributos son cuatro:

PAPER.– Es el color de los cuadritos que rodean la propia figura del carácter y que hemos considerado como apagados. En otras palabras, con PAPER definimos el color de fondo de la malla de 8 x 8. Como ya se vió, su color inicial es blanco.

INK.– Es el color de los cuadritos que definen la figura del carácter y que hemos considerado como encendidos. Su color inicial es negro.

Cuando conectamos el Spectrum escribimos en negro sobre fondo blanco.

BRIGHT.– Cada malla de 8 x 8 cuadritos puede mostrar su contenido con un brillo normal, como sucede al conectar la máquina, o con sobrebrillo, lo cual se obtiene con BRIGHT 1.

FLASH.– Cada malla de 8 x 8 cuadritos puede mostrar su contenido manteniendo el color de los cuadritos en sus colores INK y PAPER o, gracias a FLASH 1, convirtiendo alternativamente los cuadritos en color INK, en cuadritos color PAPER y al revés.

Quede claro pues, que estos atributos están referidos exclusivamente a cada malla de carácter, por lo cual no podemos manipular cada cuadrito que forma la malla de 8 x 8 que la componen. No obstante, sí podemos asignar atributos a cada una de las 768 posiciones de carácter (24 líneas x 32 columnas, si consideramos las dos líneas últimas reservadas a los mensajes del ordenador) los cuales quedarán guardados en el "**área de atributos** de la memoria RAM".

Como se puede ver en la página 165 del manual, el **"área de atributos"** del mapa de memoria permite almacenar 768 bytes; un byte por cada posición de carácter.

La forma en que cada byte de esta zona de memoria controla los atributos de cada posición de carácter es la siguiente: los tres primeros bits (del 0 al 2) controlan el color de INK, los tres siguientes (del 3 al 5) el color de PAPER, el siguiente (6) el BRIGHT y el último (7) controla el parpadeo (FLASH). Más tarde volveremos sobre esto, y especialmente sobre como utilizar la función ATTR.

Todos los atributos pueden ser usados, como ya vimos, dentro de una instrucción PRINT, en cuyo caso son definidos como **atributos temporales** ya que, una vez ejecutada la instrucción, los atributos vuelven a las condiciones que imperaban con anterioridad. Por el contrario, cuando los atributos son fijados mediante instrucciones cuya primera palabra sea una de las correspondientes a los atributos, entonces los atributos son considerados como **permanentes.**

En el caso de que deseemos efectuar una impresión sobre una o varias posiciones de carácter, pero sin alterar todos o algunos de los atributos que tengan fijados, bastará con usar el número 8. Así, si queremos realizar una impresión que mantenga la misma situación de brillo (normal o sobreiluminada) que tuviera esa posición de carácter, sólo deberemos introducir BRIGHT 8. Claro es que, si conocemos que en esa posición (o posiciones) de carácter hay una situación de sobreiluminación, sólo tendríamos que haber usado BRIGHT 1. En otras palabras, usaremos el 8 cuando no conozcamos la situación que afecta a los atributos de esa posición de carácter y se quieran mantener.

En otro orden de cosas puede ocurrir que, sin conocer el color de PAPER del que actualmente sea el fondo, se necesite un color de INK con suficiente contraste como para poder ser leído. En este momento se puede utilizar el color INK 9- con lo cual obtendremos un color de INK que, al menos, nos deje entender lo impreso. De igual forma, si no conocemos el color de INK y necesitamos un fondo que nos dé suficiente contraste, usaremos PAPER 9. Como el color 9 lo suministra el computador de acuerdo con las anteriores premisas, no siempre se obtendrá el mejor contraste pero sí, al menos, el mínimo necesario.

Nada más lejos de la casualidad que un computador. En él, todo está previsto. No obstante para multitud de aplicaciones es absolutamente necesario "manejar la casualidad". Introducir la incertidumbre.

Cuando tiramos una moneda al aire, existen dos posibilidades: obtener una cara o una cruz. De antemano no sabemos qué va a resultar. Algo similar sucede en el lanzamiento de dados o el girar de una ruleta.

Cabe pensar que con un computador no es posible llegar a situaciones imprevistas, dado que sólo se pueden obtener resultados partiendo de funciones preestablecidas.

En sentido estricto esto es cierto. Con un computador no se pueden pretender situaciones de puro azar. Todo debe estar sometido a las matemáticas. No obstante, apoyándonos en ellas conseguiremos números impredecibles desde el punto de vista práctico, los cuales son calculados a partir de una fórmula compleja y con tal rapidez, que es imposible alcanzar el resultado antes que la máquina.

Estos números que no son puramente casuales se denominan **pseudoaleatorios** y son los que producen los ordenadores a través de la función **RND**.

Con RND obtendremos números iguales o mayores que 0 y menores que 1 pero, con unos sencillos artificios, podemos obtener cualquier otro rango de variaciones. Por ejemplo:

Números comprendidos entre 1 y 10 se obtienen aleatoriamente multiplicando RND por 10, extrayendo su parte entera y sumándole uno:

```
PRINT INT (RND * 10) + 1
```

RND produce la misma serie de respuestas cada vez que se conecta la máquina. Esta característica puede ser útil para contrastar el comportamiento de distintos modelos ante la misma gama de posibilidades.

Para conseguir una sensación de casualidad total el BASIC nos provee de la sentencia **RAND** (abreviación de RANDOMIZE).

La función RAND actúa como puntero, indicando a RND donde debe iniciar su serie.

Con RAND 0 se obtiene la máxima "casualidad" posible, ya que RND iniciará su serie de acuerdo con el tiempo que lleve conectado el Spectrum. En función de esto, la rutina que nos proveerá de resultados aleatorios será:

```
10 RANDOMIZE 0
20 PRINT RND
30 GO TO 20
```

18

Si quisiéramos jugar a "cara o cruz" con una moneda o, lo que es igual, estudiar como se agrupan los resultados de un suceso dicotómico, podríamos aplicar un programa de este tipo:

```
   10 RANDOMIZE 0
   20 LET x=INT (RND*2+1)
   30 IF x=1 THEN PRINT "caras":
GO TO 20
   40 PRINT "cruces"
   50 GO TO 20
```

19

En el caso que se deseen analizar los resultados producidos en una serie de lanzamientos de un dado se podría utilizar una rutina como la que sigue:

```
10 RANDOMIZE 0
20 LET x=INT (RND*6+1)
30 PRINT x
40 GO TO 20
```

20

Este mismo caso, pero aplicado a un juego, requeriría algo de movimiento y, desde luego, más emoción. Probemos este otro:

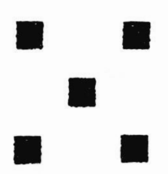
```
    2 PAPER 6: INK 1: BORDER 1
    5 GO TO 340
   10 PRINT AT 0,0;"       ";AT 1
,0;"       ";AT 2,0;"       ";AT
 3,0;"   ■   ";AT 4,0;"       ";
AT 5,0;"       ";AT 6,0;"
"
   15 RETURN
   20 GO SUB 10
   25 PRINT AT 3,0;" ■   ■ "
   26 RETURN
   30 GO SUB 20
   35 PRINT AT 3,3;"■"
   37 RETURN
   40 GO SUB 10
   45 PRINT AT 1,0;" ■   ■ ";AT 5
,0;" ■   ■ ";AT 3,3;" "
   46 RETURN
```

21

```
  50 GO SUB 40
  55 PRINT AT 3,3;"█"
  56 RETURN
  60 GO SUB 50
  65 PRINT AT 1,3;"█";AT 5,3;"█"
;AT 3,3;" "
  66 RETURN
 260 RANDOMIZE 0
 270 LET x=INT (RND*6+1)*10
 280 FOR i=1 TO 6
 290 GO SUB i*10
 300 FOR j=1 TO i
 305 BEEP .01,1
 310 NEXT j
 320 NEXT i
 330 GO SUB x
 340 INPUT "otra tirada? s/n";a$
 350 IF a$<>"s" THEN PRINT AT 19
,0;"hasta pronto": STOP
 360 GO TO 270
```

21

En este programa es de notar la interrelación existente entre los números de las líneas de programa y las subrutinas para generar las caras de un imaginario dado que nos daría una impresión de este tipo:



Así, la línea 10 nos dibuja un carácter negro para representar la cara del "uno", la línea 20 la cara del "dos", la línea 30 la del "tres" y así hasta la línea 60, que nos dará la cara del "seis". En las líneas 260 y 270 se genera un número aleatorio comprendido entre 10 y 60 que, con la instrucción de la línea 330, nos está dando los valores del dado en cada jugada.

Finalmente, en la rutina comprendida entre 280 y 320, se simula la caída del dado en el sentido de mostrarnos varias caras del dado antes de darnos la definitiva y, de esta forma, darle suspense al juego.

Observe que algunas caras del dado no reproducen exactamente las caras de un dado normal. Lo hemos hecho así para no complicar excesivamente el programa.

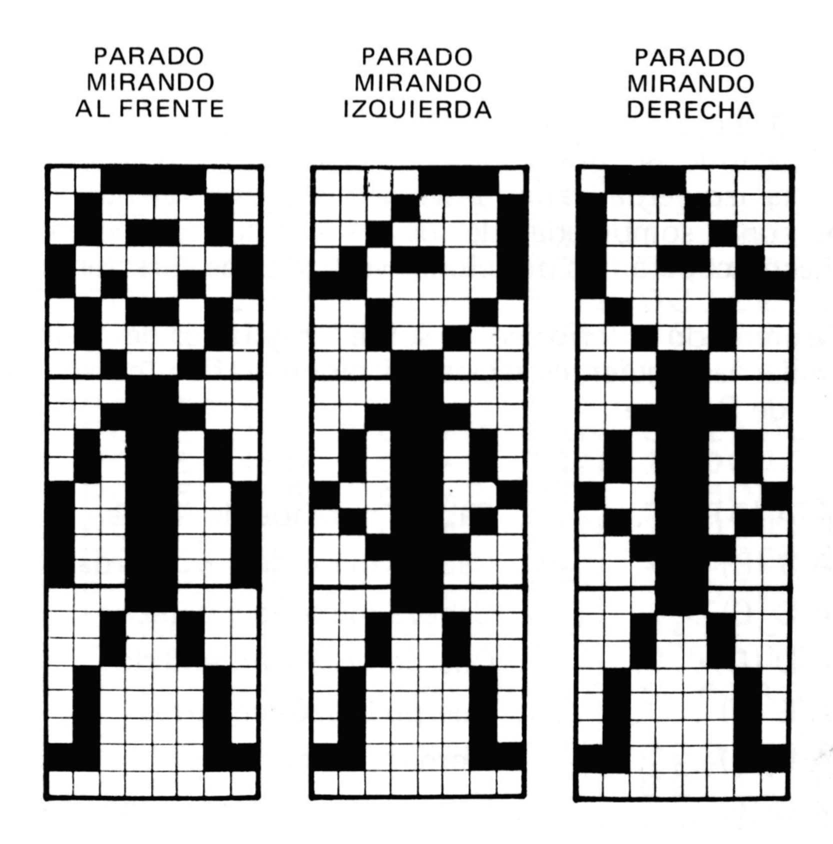
El título de este epígrafe es menos extraño de lo que parece. D. Pepe es el simpático protagonista de nuestras aventuras y, a semejanza de Geppeto y su Pinocho, vamos a darle "vida". En la medida que lo consigamos habremos dominado el movimiento de gráficos en pantalla.

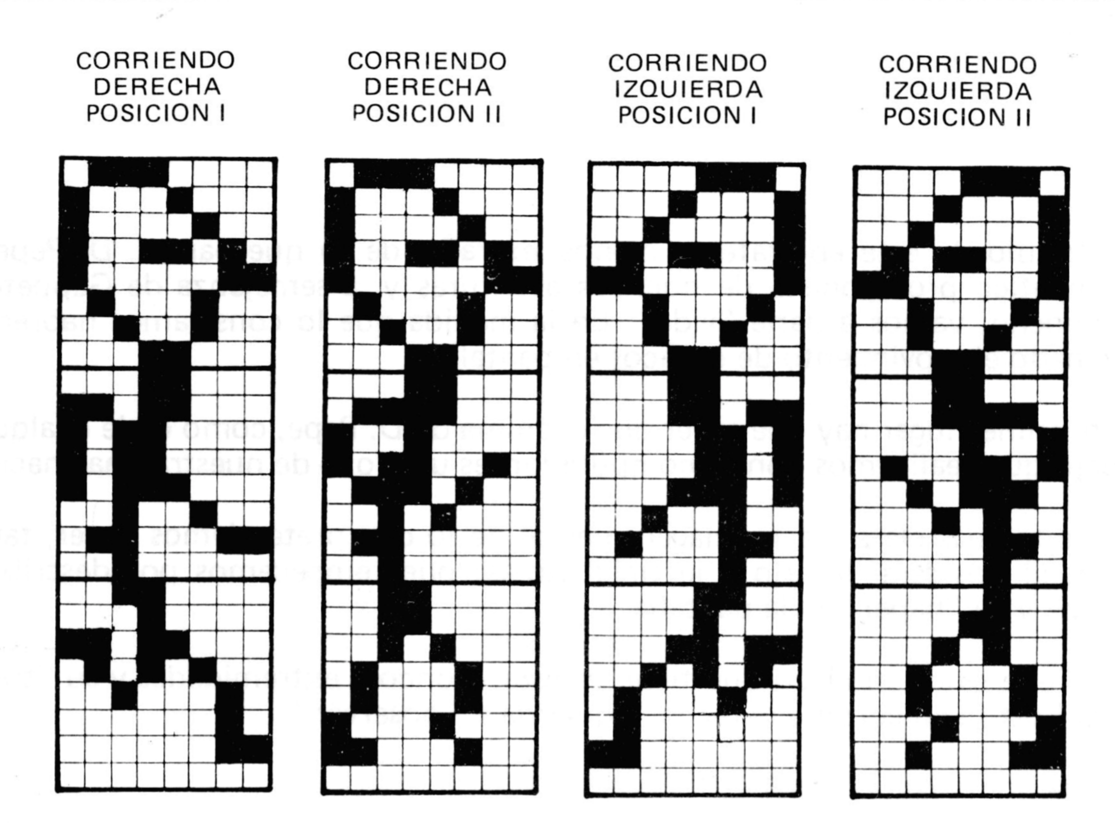
En primer lugar hay que saber que el ánima de D. Pepe, como el de cualquier trabajo que realicemos con el computador, es un trozo de nuestra imaginación.

Cuanto más clara sea la imagen mental de lo que pretendemos hacer, tanto más fácil nos será llevarlo a la práctica; así pues empecemos por describir a D. Pepe y su "papel" en la pantalla.

D. Pepe es un muñeco que tiene cabeza, tronco y extremidades y su "trabajo" consistirá en correr a derecha, izquierda y "observar".

Hagamos unos bocetos de D. Pepe:





Cuando D. Pepe esté parado moverá la cabeza a derecha, al frente y a la izquierda, adoptando las posiciones de cabeza de "corriendo a la derecha" y "corriendo a la izquierda" alternativamente y extendiendo los brazos, según muestran las zonas sombreadas de los gráficos correspondientes. Como se ve, nuestro muñeco ocupará tres posiciones verticales de carácter en pantalla.

Según lo aprendido anteriormente sobre los gráficos definidos por el usuario, vamos a asimilar las siguientes teclas, en modo gráfico (cursor en G), a diferentes "trocitos" de D. Pepe:

TECLA A (A) .... Cabeza mirando de frente
TECLA B (B) .... Cabeza mirando a izquierda
TECLA C (C) .... Cabeza mirando a derecha
TECLA D (D) .... Cuerpo brazos extendidos
TECLA E (E) .... Cuerpo brazos en jarras
TECLA F (F) .... Piernas parado
TECLA G (G) .... Piernas corriendo derecha-Posición I

TECLA H (H) . . . . Piernas corriendo izquierda-Posición I
TECLA I (I) . . . . Piernas corriendo derecha-Posición II
TECLA J (J) . . . . Piernas corriendo izquierda-Posición II
TECLA K (K) . . . . Cuerpo corriendo derecha-Posición I
TECLA L (L) . . . . Cuerpo corriendo derecha-Posición II
TECLA M (M) . . . . Cuerpo corriendo izquierda-Posición I
TECLA N (N) . . . . Cuerpo corriendo izquierda-Posición II

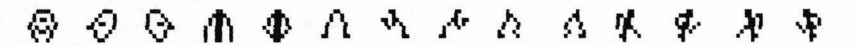
Las rutinas que nos darían las posiciones de D. Pepe, según lo ya estudiado, serían las siguientes:

```
   9 REM ***cabeza mirando de fr
ente***
  10 POKE USR "a"+0,BIN 00111100
  20 POKE USR "a"+1,BIN 01000010
  30 POKE USR "a"+2,BIN 01011010
  40 POKE USR "a"+3,BIN 10000001
  50 POKE USR "a"+4,BIN 10100101
  60 POKE USR "a"+5,BIN 01011010
  70 POKE USR "a"+6,BIN 01000010
  80 POKE USR "a"+7,BIN 00100100
  99 REM ***cabeza mirando a la
izquierda***
 100 POKE USR "b"+0,BIN 00001110
 110 POKE USR "b"+1,BIN 00010001
 120 POKE USR "b"+2,BIN 00100001
 130 POKE USR "b"+3,BIN 01011001
 140 POKE USR "b"+4,BIN 11000001
 150 POKE USR "b"+5,BIN 00100010      22
 160 POKE USR "b"+6,BIN 00100100
 170 POKE USR "b"+7,BIN 00011000
 199 REM ***cabeza mirando a la
derecha***
 200 POKE USR "c"+0,BIN 01110000
 210 POKE USR "c"+1,BIN 10001000
 220 POKE USR "c"+2,BIN 10000100
 230 POKE USR "c"+3,BIN 10011010
 240 POKE USR "c"+4,BIN 10000011
 250 POKE USR "c"+5,BIN 01000100
 260 POKE USR "c"+6,BIN 00100100
 270 POKE USR "c"+7,BIN 00011000
 299 REM ***cuerpo brazos extend
idos***
 300 POKE USR "d"+0,BIN 00011000
 310 POKE USR "d"+1,BIN 00111100
 320 POKE USR "d"+2,BIN 01011010
 330 POKE USR "d"+3,BIN 01011010
```

```
340 POKE USR "d"+4,BIN 10011001
350 POKE USR "d"+5,BIN 10011001
360 POKE USR "d"+6,BIN 10011001
370 POKE USR "d"+7,BIN 10011001
399 REM ***cuerpo brazos en jar
ras***
400 POKE USR "e"+0,BIN 00011000
410 POKE USR "e"+1,BIN 00111100
420 POKE USR "e"+2,BIN 01011010
430 POKE USR "e"+3,BIN 01011010
440 POKE USR "e"+4,BIN 10011001
450 POKE USR "e"+5,BIN 01011010
460 POKE USR "e"+6,BIN 00111100
470 POKE USR "e"+7,BIN 00011000
499 REM ***piernas parado***
500 POKE USR "f"+0,BIN 00011000
510 POKE USR "f"+1,BIN 00100100
520 POKE USR "f"+2,BIN 00100100
530 POKE USR "f"+3,BIN 01000010
540 POKE USR "f"+4,BIN 01000010
550 POKE USR "f"+5,BIN 01000010
560 POKE USR "f"+6,BIN 11000011
570 POKE USR "f"+7,BIN 00000000
599 REM ***piernas corriendo de
recha posicion I***
600 POKE USR "g"+0,BIN 00110000
610 POKE USR "g"+1,BIN 00010000
620 POKE USR "g"+2,BIN 11011000        22
630 POKE USR "g"+3,BIN 01010100
640 POKE USR "g"+4,BIN 00100010
650 POKE USR "g"+5,BIN 00000010
660 POKE USR "g"+6,BIN 00000011
670 POKE USR "g"+7,BIN 00000000
699 REM ***piernas corriendo iz
quierda posicion I***
700 POKE USR "h"+0,BIN 00001100
710 POKE USR "h"+1,BIN 00001000
720 POKE USR "h"+2,BIN 00011011
730 POKE USR "h"+3,BIN 00101010
740 POKE USR "h"+4,BIN 01000100
750 POKE USR "h"+5,BIN 01000000
760 POKE USR "h"+6,BIN 11000000
770 POKE USR "h"+7,BIN 00000000
799 REM ***piernas corriendo de
recha posicion II***
800 POKE USR "i"+0,BIN 00100000
810 POKE USR "i"+1,BIN 00110000
820 POKE USR "i"+2,BIN 00101000
830 POKE USR "i"+3,BIN 01000100
840 POKE USR "i"+4,BIN 01000100
850 POKE USR "i"+5,BIN 10001000
860 POKE USR "i"+6,BIN 11000100
```

```
 870 POKE USR "i"+7,BIN 00000000
 899 REM ***piernas corriendo iz
quierda posicion II***
 900 POKE USR "j"+0,BIN 00000100
 910 POKE USR "j"+1,BIN 00001100
 920 POKE USR "j"+2,BIN 00010100
 930 POKE USR "j"+3,BIN 00100010
 940 POKE USR "j"+4,BIN 00100010
 950 POKE USR "j"+5,BIN 00010001
 960 POKE USR "j"+6,BIN 00100011
 970 POKE USR "j"+7,BIN 00000000
 999 REM ***cuerpo corriendo der
echa posicion I***
1000 POKE USR "k"+0,BIN 00011000
1010 POKE USR "k"+1,BIN 11011000
1020 POKE USR "k"+2,BIN 10110000
1030 POKE USR "k"+3,BIN 10110000
1040 POKE USR "k"+4,BIN 10111000
1050 POKE USR "k"+5,BIN 00100100
1060 POKE USR "k"+6,BIN 01100010
1070 POKE USR "k"+7,BIN 00100000
1099 REM ***cuerpo corriendo der
echa posicion II***
1100 POKE USR "l"+0,BIN 00011000
1110 POKE USR "l"+1,BIN 01111000
1120 POKE USR "l"+2,BIN 10010000      22
1130 POKE USR "l"+3,BIN 10110010
1140 POKE USR "l"+4,BIN 01110100
1150 POKE USR "l"+5,BIN 00101000
1160 POKE USR "l"+6,BIN 01100000
1170 POKE USR "l"+7,BIN 00100000
1199 REM ***cuerpo corriendo izq
uierda posicion I***
1200 POKE USR "m"+0,BIN 00011000
1210 POKE USR "m"+1,BIN 00011011
1220 POKE USR "m"+2,BIN 00001101
1230 POKE USR "m"+3,BIN 00001101
1240 POKE USR "m"+4,BIN 00011101
1250 POKE USR "m"+5,BIN 00100100
1260 POKE USR "m"+6,BIN 01000110
1270 POKE USR "m"+7,BIN 00000100
1299 REM ***cuerpo corriendo izq
uierda posicion II***
1300 POKE USR "n"+0,BIN 00011000
1310 POKE USR "n"+1,BIN 00011110
1320 POKE USR "n"+2,BIN 00001001
1330 POKE USR "n"+3,BIN 01001101
1340 POKE USR "n"+4,BIN 00101110
1350 POKE USR "n"+5,BIN 00010100
1360 POKE USR "n"+6,BIN 00000110
1370 POKE USR "n"+7,BIN 00000100
```

De esta forma, en las teclas convenidas, tendríamos esta serie de caracteres:



Los cuales, debidamente acoplados darán unas imágenes, como se verá más adelante, similares a éstas:



Vamos a analizar algunos conceptos básicos del movimiento antes de continuar.

Es práctico evitar, en lo posible, utilizar la alta resolución de gráficos —tema que veremos más adelante— ya que obligamos a la máquina a trabajar más, debiendo, por tanto, aprovechar todas las posibilidades que nos ofrecen los caracteres gráficos definidos —o no— por el usuario.

En otro orden de cosas sabemos que, tanto el cine como la televisión transmiten sus imágenes secuencialmente, de tal forma que nuestra vista lo interpreta como un movimiento continuo. Nosotros utilizaremos el mismo principio, apoyándonos en los medios que nos brinda el computador. Empecemos por hacer aparecer y desaparecer una imagen con una velocidad controlada:

```
1000 LET e=10
1010 LET a=60
1020 PRINT AT 0,0;"▚"
1030 FOR i=1 TO e: NEXT i          23
1040 PRINT AT 0,0;" "
1050 FOR j=1 TO a: NEXT j
1060 GO TO 1000
```

Las líneas 1000 y 1010 nos fijan tiempos de encendido **(e)** y apagado **(a)**. Los bucles 1030 y 1050 son retardos de encendido y apagado. En la línea 1040 colocamos un carácter blanco (con el cursor en modo gráfico apretar la tecla con el número 8) para producir la sensación de "apagado". Cambie Vd. los valores de **e** y **a** para observar diferentes posibilidades.

El siguiente paso para conseguir el efecto de movimiento, consistirá en hacer "aparecer" el carácter, hacerlo "desaparecer" en la misma posición, para volverlo hacer aparecer uno o varios espacios más adelante:

```
1000 PRINT AT 0,0;"▘▗"
1010 FOR j=0 TO 10: NEXT j
1020 FOR i=0 TO 30
1030 FOR j=0 TO 10: NEXT j
1040 PRINT AT 0,i;" ▘▗"
1050 NEXT i
1060 GO TO 1000
```

24

En las líneas 1000 producimos la impresión de la primera posición. En la 1020 tenemos un bucle que obligará al gráfico a recorrer toda la línea 0.

En la línea 1040 hacemos avanzar el gráfico una posición "borrando" la anterior, gracias al carácter "blanco" introducido. Con esta rutina el movimiento se produce al margen de nuestro control.

Completemos esta introducción al movimiento, desarrollando un programa que nos permita controlar el desplazamiento de un hipotético muñeco que ocupa dos posiciones de pantalla, una encima de otra, tal como éste:



y que podría haber estado, sin dificultad, compuesto por dos "trocitos" de D. Pepe, pero que por razones prácticas está formado por un asterisco arriba y otros caracteres gráficos abajo.

Con este carácter combinado vamos a lograr la sensación de que la parte inferior está moviéndose constantemente:

```
1000 LET c=15
1010 LET a$=INKEY$
1020 IF a$="5" THEN LET c=c-1
1030 IF a$="8" THEN LET c=c+1
1040 PRINT AT 0,c;"    *    ";AT 1     25
,c;"   █▀█   ";
1050 BEEP .01,10
1060 PRINT AT 0,c;"    *    ";AT 1
,c;"   █▀█   ";
1070 GO TO 1010
```

El comando BEEP, además de emitir un sonido, actua como una pausa o bucle de retardo. Dicho de otro modo, durante el tiempo que está emitiendo un BEEP, el computador no hace otra cosa y esto se puede comprobar alargando la duración de la línea 1050 del anterior programa con un BEEP 4, 10, por ejemplo.

Las normas básicas del movimiento son las expuestas anteriormente y serán de aplicación inmediata en la animación de D. Pepe, con algunas variaciones. En primer lugar, los gráficos son distintos según se desplace nuestro amigo a la derecha o a la izquierda. En segundo lugar la cabeza ha de mirar, como ya se acordó más arriba, a derecha, al frente y a la izquierda cuando esté parado y extendiendo sus brazos —desde la posición en "jarras"— alternativamente.

Veamos este último caso:

```
2000 REM ***Don Pepe***
2010 LET c=15
2020 GO SUB 2080
2030 PRINT AT 0,c;" A ";AT 1,c;"
 E ";AT 2,c;" F ";
2040 BEEP .3,15
2050 PRINT AT 0,c;" B ";AT 1,c;"      26
 D ": BEEP .3,5
2051 PRINT AT 0,c;" A ";AT 1,c;"
 E ";AT 2,c;" F ": BEEP .3,15
2052 PRINT AT 0,c;" C ";AT 1,c;"
 D "
2060 BEEP .3,5
2070 GO TO 2020
```

Este programa no se puede correr.

Doy por supuesto, que las rutinas que generan los caracteres gráficos de los distintos "trocitos" de D. Pepe ya están en memoria.

El programa que nos determinará a D. Pepe corriendo a izquierda o derecha y nos permitirá controlar el movimiento, se basa en todo lo estudiado anteriormente:

```
2080 LET a$=INKEY$
2090 IF a$="5" THEN LET c=c-1: G
O SUB 2120: GO TO 2080
2100 IF a$="8" THEN LET c=c+1: G
O SUB 2180: GO TO 2080
2110 RETURN
2120 IF c<0 THEN LET c=30: PRINT
 AT 0,1;" ";AT 1,1;" ";AT 2,1;"
"
2130 PRINT AT 0,c;" B ";AT 1,c;"
 M ";AT 2,c;" H "
2140 BEEP .1,15
2150 PRINT AT 0,c;" B ";AT 1,c;"
 N ";AT 2,c;" J "
2160 BEEP .1,20
2170 RETURN
2180 IF c>30 THEN LET c=0: PRINT
 AT 0,31;" ";AT 1,31;" ";AT 2,31
;" "
2190 PRINT AT 0,c;" C";AT 1,c;"
K";AT 2,c;" G"
2200 BEEP .1,15
2210 PRINT AT 0,c;" C ";AT 1,c;"
 L ";AT 2,c;" I "
2220 BEEP .1,20
2230 RETURN
```

27

Este programa no se puede correr tal y como está. Para conseguir el efecto de movimiento de Don Pepe, será necesario correr juntos los programas 22, 26 y 27.

A otro nivel de complejidad se llega al introducir más de un movimiento simultáneo.

Los rudimentos de estos programas se basan en la lectura continua y secuencial de las condiciones que se le impongan a cada movimiento.

Para ver esto supongamos dos asteriscos que se han de mover por la misma línea y con una velocidad, de uno con respecto a otro, del doble. Esta última condición implica que mientras un asterisco avanza un espacio el otro avanza dos.

Veamos este ejemplo:

```
10 LET c=0
20 LET c1=0
30 LET v=1: LET v1=2
40 GO SUB 70
50 GO SUB 120
60 GO TO 40
70 PRINT AT 0,c;" "
80 LET c=c+v
```

28

```
  90 IF c=31 OR c=0 THEN LET v=-
v
 100 PRINT AT 0,c;"*"
 110 RETURN
 120 PRINT AT 5,c1;" "
 130 LET c1=c1+v1
 140 IF c1>31 THEN LET c1=0
 150 PRINT AT 5,c1;"*"
 160 RETURN
```

28

En la línea 30 fijamos las velocidades de cada asterisco. En la 40 dirigimos el programa a la subrutina de avance del más lento. En la 50 a la de avance del más rápido. En la línea 90 invertimos la marcha del asterisco más lento al llegar a los límites de la pantalla.

En la 140 determinamos que el asterisco más rápido al llegar al límite izquierdo de la pantalla, vuelva a comenzar.

Gracias a la línea 90 vemos "rebotar" el asterisco en los bordes izquierdo y derecho de la pantalla, moviéndose siempre en la misma línea.

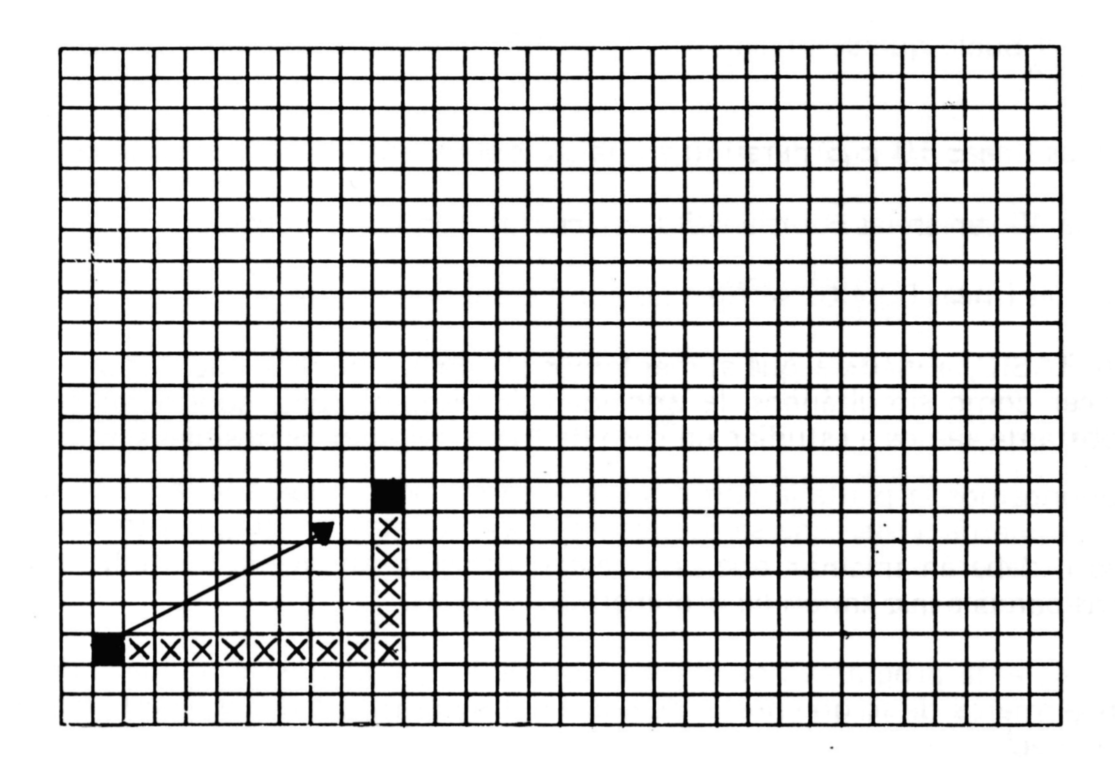
En el caso de balones o pelotas rebotando contra paredes, las trayectorias ya no tienen porqué ser horizontales o verticales y, por tanto, requieren un tratamiento especial.

Una pelota en movimiento de un punto cualquiera a otro y sometida a una velocidad **(v)**, se puede descomponer en una velocidad horizontal **(h)** y en una vertical **(v)**. En otras palabras, cada desplazamiento de la pelota requiere moverla un número de espacios horizontales y otro (igual o distinto) verticales. (Ver figura en página 41).

Si llamamos **l** a la coordenada que repreesenta a las líneas y **c** a la que representa a las columnas. deberemos añadir la velocidad horizontal (h) a la coordenada **c** y la velocidad vertical (v) a la **l**.

Por otra parte resulta evidente que, cuando una pelota "choca" con una pared vertical, devuelve la velocidad horizontal en sentido contrario, ya que la vertical queda invariable.

Cuando "choca" con una pared horizontal, por similar razón, sólo devuelve la vertical.



Si aplicamos estos conceptos veremos a la pelota rebotar, ausente de efectos tales como el rozamiento, la gravedad, etc. Estas cuestiones se verán más adelate..

De momento estudiemos este programa:

```
   5 GO SUB 80
  10 LET l=1: LET c=1
  20 LET v=1: LET h=1
  30 PRINT AT l,c;" "
  40 LET l=l+v: LET c=c+h
  50 IF l=1 OR l=20 THEN LET v=-
v: BEEP .3,15
  60 IF c=1 OR c=30 THEN LET h=-
h: BEEP .3,20
  70 PRINT AT l,c;"■"
  72 PAUSE 5
  75 GO TO 30
  80 FOR i=0 TO 31
  90 PRINT AT 0,i;"■";
 100 PRINT AT 21,i;"■";
 110 IF i>21 THEN GO TO 140
 120 PRINT AT i,0;"■";
 130 PRINT AT i,31;"■";
 140 NEXT i
 150 RETURN
```

29

Con el bucle que se inicia en la línea 80, dibujamos el marco de la zona de "rebotes".

En las líneas 10 y 20 determinamos las coordenadas iniciales y las velocidades.

En la 40 varían las coordenadas de acuerdo con las velocidades.

Con las líneas 50 y 60 se fijan las condiciones de rebote.

En BASIC, a medida que aumentamos el número de movimientos que deban aparecer como simultáneos, la velocidad de respuesta baja considerablemente. No obstante vamos a estudiar un caso de tres movimientos posibles simultáneos.

Supongamos la pantalla como un recinto cerrado y que a lo largo de sus paredes izquierda y derecha se deslizan sendas raquetas, las que, controladas por teclado, deberán impedir que una pelota, moviéndose por la pantalla, choque con las paredes que tratan de proteger las raquetas.

El siguiente programa nos da una idea de lo que puede ser un juego de tenis o, en términos de lo que pretendemos, un movimiento triple y, aparentemente, simultáneo:

```
   2 BORDER 1: INK 1: PAPER 6
   5 REM ***recuadro de juego***
  10 FOR i=0 TO 31
  20 PRINT INK 7;AT 0,i;"█";AT 2
1,i;"█"
  50 NEXT i
  52 PAUSE 50
  55 REM **coordenadas iniciales
 de la pelota***
  60 LET l=5: LET c=15
  65 REM ***velocidad inicial de
 la pelota***
  70 LET v=1: LET h=1
  75 REM ***coordenadas de raque        30
ta derecha***
  80 LET lrd=10: LET crd=31
  85 REM ***coordenadas de raque
ta izquierda***
  90 LET lri=10: LET cri=0
  95 REM ***trayectoria pelota**
*
 100 GO SUB 130
 105 REM ***desplazamiento raque
ta derecha***
 110 GO SUB 400
 115 REM ***desplazamiento raque
ta izquierda***
```

```
 120 GO SUB 500
 122 REM ***cierra el proceso y
lo repite***
 125 GO SUB 100
 130 IF l=1 OR l=20 THEN LET v=-
v: BEEP .3,15
 135 IF c=0 OR c=31 THEN CLS : G
O TO 5
 140 IF c=1 THEN GO SUB 300
 150 IF c=30 THEN GO SUB 200
 160 PRINT INK 1;AT l,c;" "
 170 LET l=l+v: LET c=c+h
 180 PRINT AT l,c;"█"
 190 RETURN
 195 REM *********************
 200 LET dD=l-lrd
 210 IF dD<0 OR dD>5 THEN BEEP .
1,0: RETURN
 220 LET h=-h                            30
 230 RETURN
 300 LET dI=l-lri
 310 IF dI<0 OR dI>5 THEN BEEP .
1,0: RETURN
 320 LET h=-h
 330 RETURN
 340 REM *********************
 400 LET a$=INKEY$
 410 IF a$="0" AND lrd>1 THEN LE
T lrd=lrd-1: GO TO 430
 420 IF a$="p" AND lrd<17 THEN L
ET lrd=lrd+1
 430 PRINT AT lrd,31;" ";AT lrd+
1,31;"█";AT lrd+2,31;"█";AT lrd+
3,31;"█";AT lrd+4,31;" "
 440 RETURN
 500 LET b$=INKEY$
 510 IF b$="1" AND lri>1 THEN LE
T lri=lri-1: GO TO 530
 520 IF b$="q" AND lri<17 THEN L
ET lri=lri+1
 530 PRINT AT lri,0;" ";AT lri+1
,0;"█";AT lri+2,0;"█";AT lri+3,0
;"█";AT lri+4,0;" "
 540 RETURN
```

Este programa requiere pocos comentarios, ya que los REM introducidos, junto con todo lo visto hasta el momento, lo hacen innecesario. Analice el listado del programa y, además de aumentar su capacidad de análisis, averiguará cómo manejar las raquetas, el movimiento de la pelota, etc.

Unas consideraciones finales ayudarán en el diseño de programas. En primer lugar la programación en BASIC, como se ha podido ver, no presenta especial dificultad pero, a veces, los resultados obtenidos aparecen como lentos debido a que el Spectrum traduce a lenguaje máquina, una a una, todas las instrucciones de un programa; este tipo de "traducción" hace que llamemos **intérprete** a esta forma de BASIC, en contraposición al **compilador**, el cual traduce el programa completo a lenguaje máquina antes de correrlo dando, por esto, una velocidad de proceso mucho mayor.

Otro motivo de "lentitud" procede de los saltos incondicionales tales como GOTO o GOSUB, que obligan a continuar el programa en una línea determinada, pero, como la localización de esta línea implica una búsqueda secuencial desde el principio, siempre hay una pérdida añadida de tiempo. De aquí debe seguirse que una adecuada ubicación de subrutinas y funciones ayudará a mejorar la velocidad de ejecución de los programas.

En todo caso, antes de llegar a diseñar un programa que sea realmente operativo, habrá que hacer modificaciones y solventar errores.

Cuando aparece un error en un programa bien estructurado es relativamente fácil localizarlo y aplicar el remedio oportuno. Para conseguir una buena estructuración se debe tratar fundamentalmente:

1°.– Definir con claridad las variables y tratar de agruparlas por tipos. Por ejemplo: V, V1, V2, ... v, v1, v2, para referirnos a las velocidades.

2°.– Colocar los buches FOR/NEXT en una sola línea. Si tal cosa no fuera posible, entonces colocar las sentencias FOR y NEXT al principio de su línea, de forma que sean fácilmente identificables.

3°.– Las sentencias REM son muy útiles para clarificar un programa, pero no conviene abusar de ellas pues restan velocidad.

4°.– Siempre que pueda escriba programas cortos que recurran a tantas subrutinas como necesite, de forma que los errores y modificaciones sean fáciles de poner en orden. Obviamente, aquí las subrutinas tienen una misión clarificadora y no, necesariamente, son introducidas porque vayan a ser utilizadas muchas veces.

5°.– Numerar el programa de acuerdo con grupos de actividades. Por ejemplo: hasta la línea 500 fijar parámetros, de la 1000 a la 2000 colocar el programa principal, etc.

Una vez estructurado el programa, si éste —al hacerlo correr— no funciona adecuadamente, deberá ser comprobado para encontrar los motivos. Algunas de las normas básicas del proceso de comprobación pueden ser:

1°.— Introducir eventuales sentencias de STOP hasta ahorquillar el problema. Con CONT podrá hacer seguir el programa con su rutina normal.

2°.— Introducir condicionantes del tipo IF-THEN STOP para analizar, por ejemplo, si las variables llegan con un valor adecuado a determinadas líneas de programa.

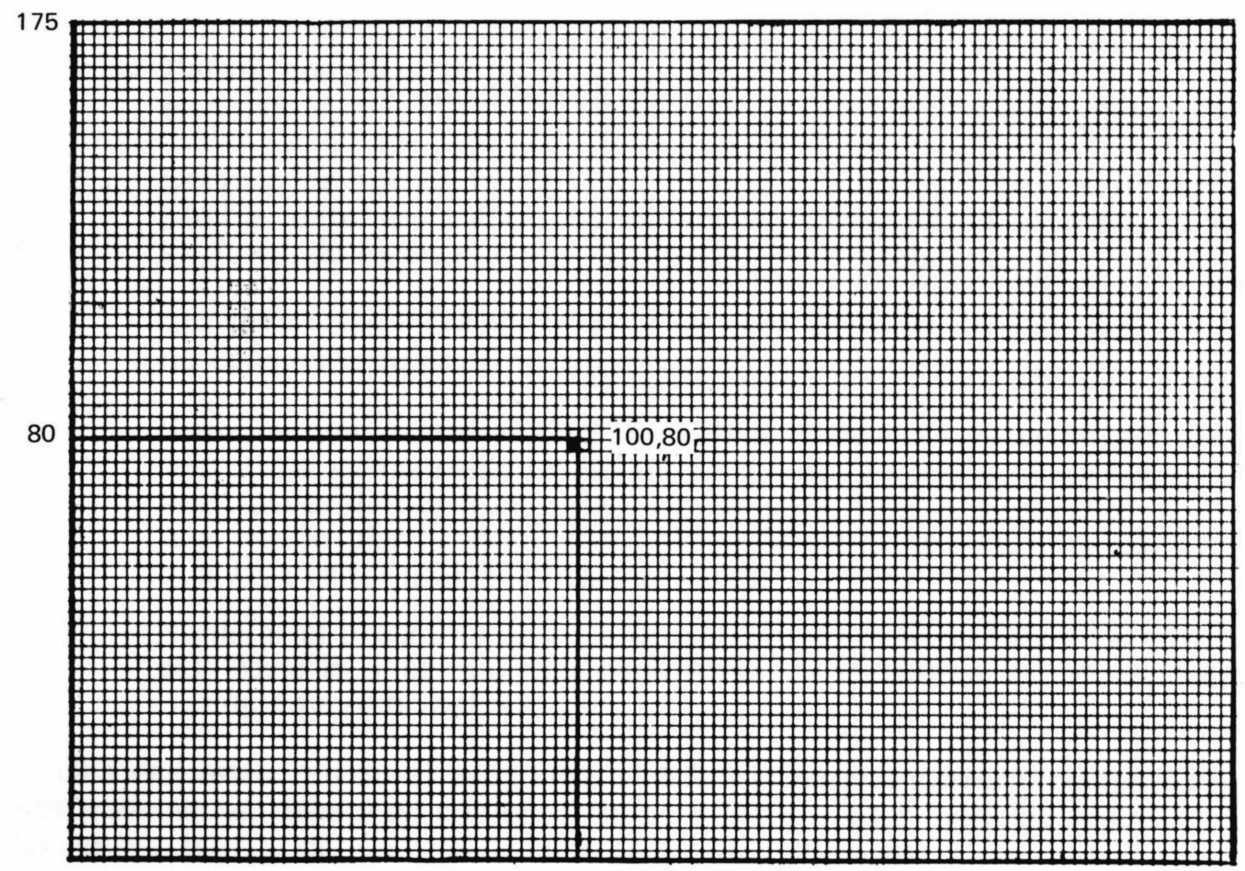
3°.— Si presume que algún bloque de su programa no funciona según lo previsto: ¡Anúlelo! Bien por borrado, bien con un salto incondicional del tipo GOTO.

4°.— Introduzca ocasionales PRINT de variables para conocer los valores con que opera el programa en un sector, seguido, si necesita reflexionar sobre ellos, de un STOP.

Evidentemente, una vez localizado y subsanado el fallo, deberá eliminar todas las instrucciones suplementarias introducidas.

Como ya se dijo anteriormente, trabajar en alta resolución de gráficos implica hacer trabajar al ordenador bastante más que cuando actúa y controla caracteres o, lo que es igual, manejar mallas completas de 8 x 8 cuadritos. Evidentemente, actuando sobre caracteres —el Spectrum— debe manejar 704 posiciones posibles de carácter, mientras que en alta resolución pasaría a controlar 45.086 cuadritos.

Como ya sabemos, con una instrucción PRINT AT (línea, columna) posicionaremos un grupo de 64 cuadritos (una malla de 8 x 8), mientras que, con una sentencia PLOT, situaremos exclusivamente un cuadrito. Estos cuadritos son la mínima porción de pantalla que el Spectrum puede controlar y que en inglés son conocidos como "**pixel**". Cualesquiera de estos cuadritos, estará situado en un plano coordenado cuyo eje de abcisas (horizontal) varía entre 0 y 255 y el de ordenadas (vertical) entre 0 y 175. Ambos ejes comienzan la numeración en el ángulo inferior izquierdo de la pantalla:



El cuadrito 100, 80 estará situado en la columna 100, fila 80. Obsérvese que PLOT, a diferencia de PRINT, considera primero la columna y después la fila y, muy importante, que la numeración comienza en el ángulo inferior izquierdo de la pantalla. Estas diferencias serán causa de no pocas dificultades cuando aparezcan ambas definiciones —alta y baja— en un mismo programa.

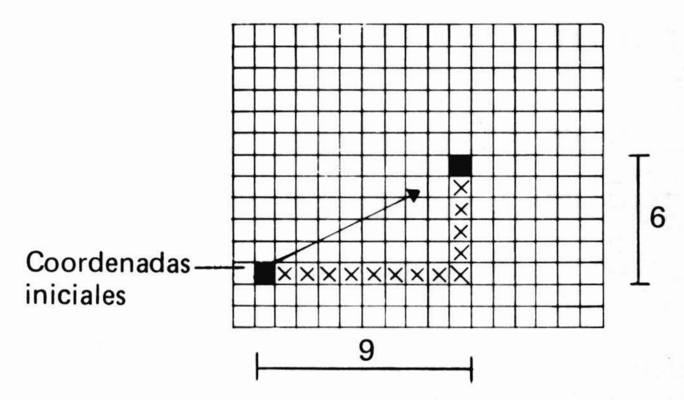
Estudiaremos a continuación los tres comandos del Spectrum para alta resolución de gráficos:

PLOT **DRAW** **CIRCLE**

Con **PLOT (x, y)** encenderemos —ponemos en color INK— el cuadrito situado en la columna **x**, fila **y**. Tanto x como y pueden ser números o cualquier expresión que finalmente den unos valores comprendidos entre 0 y 255, y 0 y 175 respectivamente.

DRAW x, y nos traza una recta desde el lugar donde estuviera situado el último cuadrito que hubieramos manejado hasta otro cuadrito alejado del primero según los parámetros x, y que siguen a DRAW.

Es decir, si Vd. quiere trazar una recta desde una posición cualquiera hasta otra que esté, por ejemplo, 9 cuadritos horizontales a la derecha y 6 cuadritos verticales hacia arriba:



deberá teclear DRAW 9,6, bien entendido que si el cuadrito final está a la izquierda del inicial, deberemos colocar el signo - para indicar esta circunstancia. En el mismo orden de cosas habría que actuar para desplazar la recta en sentido vertical y hacia abajo.

Resumen de signos.

Dado un punto PLOT x, y inicial, las coordenadas del final de la recta deberán darse con signo positivo o negativo, según la dirección y el sentido de la recta.

Recta   Coordenadas (final de la recta)

Recta	x	y
↗	+	+
↘	+	—
↙	—	—
↖	—	+
↑	Ø	+
→	+	Ø
↓	Ø	—
←	—	Ø

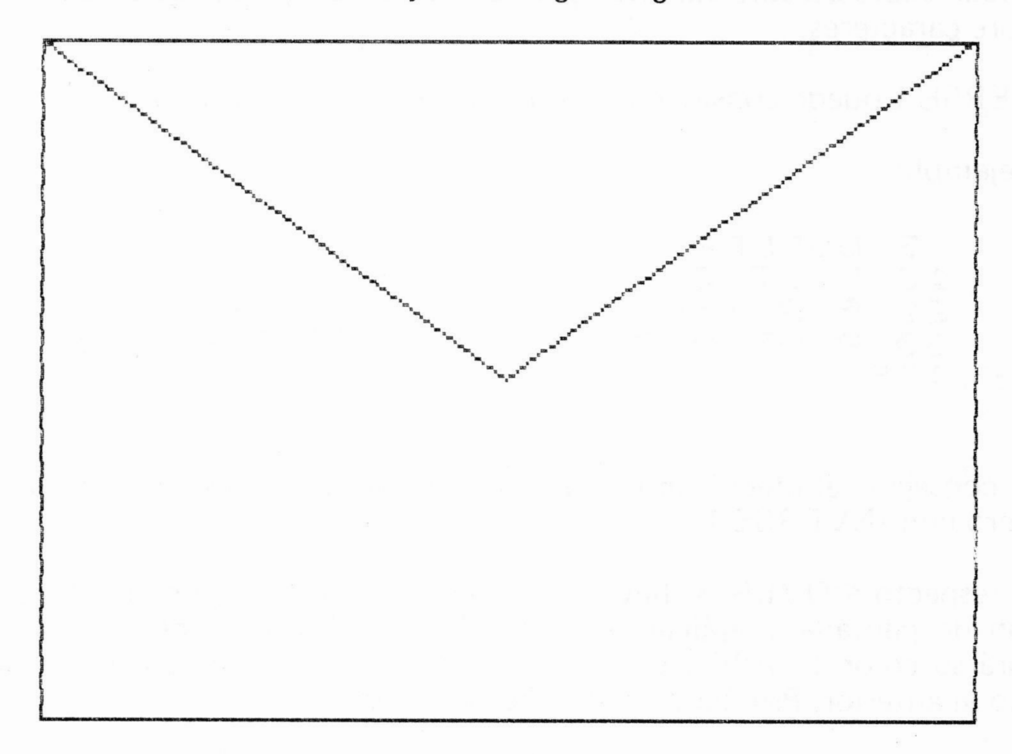
```
10 INPUT "abscisa del cuadrito ";x
20 INPUT "ordenada del cuadrito  ";y
30 PLOT x,y
40 INPUT "abscisa cuadrito final  ";x1
50 INPUT "ordenada cuadrito final  ";y1
60 DRAW x1,y1
70 GO TO 40
```

31

Con DRAW podemos trazar líneas quebradas, comenzando cada segmento donde acaba el anterior. Esta posibilidad se puede observar mediante el programa anterior:

Pruebe a iniciar el programa (líneas 10 y 20) con x = 0, y = 167. Continue con x = 255, y = 0; despues x = 0, y =-167. Siga con x = -255, y = 0 y finalice con x = 0, y = 167.

Para acabar este ejercicio, haga x = 127, y = – 83 y después x = 127, y = 83. Y de esta forma se obtendrá el dibujo de la figura siguiente:

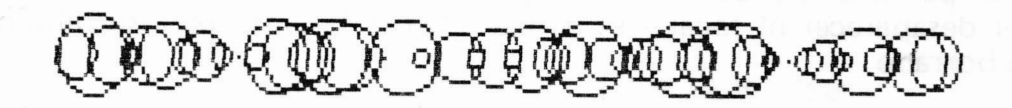


Para trazar una circunferencia con centro en el punto **x, y,** con radio **r**, bastará con utilizar **CIRCLE x, y, r.** Para comprobar esta función de forma inmediata apriete CIRCLE y a continuación 133, 88, 50. Ahora ENTER. Corra el programa.

```
 5 BORDER 1: INK 1: PAPER 6
10 FOR x=10 TO 245 STEP 5
20 LET r=RND*10
30 CIRCLE x,88,r
35 IF x>70 THEN GO TO 40
40 NEXT x
```

32

y obtendremos una sucesión de circunferencias de radios aleatorios, comprendidos entre 0 y 10, similar a esta:



Tanto INVERSE como OVER pueden ser usados en alta resolución, ya que sólo actúan sobre un solo cuadrito y su forma de trabajar es bastante más simple que sobre caracteres.

INVERSE 1 puede considerarse como un sistema de borrado.

Por ejemplo:

```
  5 BORDER 1: INK 1: PAPER 6
 10 PLOT 0,0: DRAW 255,175
 20 FOR i=0 TO 175: NEXT i
 30 PLOT 0,0: DRAW INVERSE 1,25
5,175
```

33

Para conseguir el efecto de borrado se debe seguir la misma dirección de trazado pero con INVERSE 1.

Con respecto a OVER, si hay un puntillo en color INK en una determinada posición de pantalla y aplicamos un OVER 1 sobre esa posición, el puntillo cambiará su color a PAPER y al revés. El siguiente programa cambia algo con respecto al anterior. Pruebe y analice las variaciones.

```
  5 BORDER 1: INK 1: PAPER 6
 10 PLOT 0,0: DRAW 255,175
 20 FOR i=0 TO 175: NEXT i
 30 PLOT 0,0: DRAW OVER 1,255,1
75
 40 PAUSE 50
 50 PLOT 0,0: DRAW OVER 0,255,1
75
 60 GO TO 10
```

34

Si combinamos OVER 1 con INVERSE 1, ambos efectos se cancelan, ya que INVERSE 1 hace a todos los comandos de alta resolución donde se aplique, producir cuadritos en color PAPER. Según lo visto anteriormente, con OVER 1 un puntillo en color PAPER pasa a tomar el color INK. Por tanto nada cambió.

Una vez aquí, es conveniente resaltar un par de cosas. Primero, para trazar figuras en pantallas donde ya exista alguna impresión previa —la cual no se quiera hacer desaparecer ni tocar— es de uso OVER 1, tanto para su dibujo como para su borrado.

Por ejemplo:

```
  5 BORDER 1: INK 1: PAPER 6
 10 PRINT AT 10,5;"Los colores
del Spectrum"
 20 PLOT 0,0: DRAW OVER 1;255,1
75
 30 PAUSE 75
 40 PLOT 0,0: DRAW OVER 1;255,1
75
```
35

En segundo lugar, una instrucción del tipo PLOT INVERSE 1, OVER 1, x, y será una forma, a veces muy útil, de posicionar el cursor de los pixels "invisiblemente" donde nos interese.

POINT es otra función que actúa sobre cuadritos individuales y su misión es indicarnos si ese cuadrito está encendido o apagado (INK o PAPER), respondiendo 0 para apagado (color PAPER) o 1 para encendido (color INK).

Ejemplo:

```
  5 BORDER 1: INK 1: PAPER 6
 10 PRINT AT 20,1;"■"
```
36

Al correr este programa habremos puesto en color INK una malla de 8 x 8 cuadritos, en la posición de carácter situada en la línea 20, columna 1, o lo que es igual, todos los cuadritos situados entre las abcisas 8 y 15, y entre las ordenadas 8 y 15 (Mire el desplegable de la página 17).

Si en estas condiciones probamos, por ejemplo, la sentencia directa PRINT POINT (12, 12), obtendremos un 1 (encendido) como respuesta del ordenador. Si por el contrario, y una vez corrido el programa nuevamente, tecleamos PRINT POINT (20, 20) la respuesta sería Ø, ya que el "pixel" elegido está fuera del carácter azul (INK 1) impreso por nuestro programa.

Puesto que los atributos controlan una posición de carácter completa (malla de 8 x 8 cuadritos), nunca afectarán a un sólo cuadrito individual, lo cual traerá algunas complicaciones que se deben conocer. Así, mientras podemos usar INK, PAPER, BRIGHT y FLASH dentro de sentencias de alta resolución, los atributos siempre actuarán sobre toda la malla de la posición de carácter donde los cuadritos controlados por PLOT, DRAW o CIRCLE estuvieran. Dicho de otra forma, dentro de una posición de carácter no pueden existir más de dos colores: el correspondiente a INK y el correspondiente a PAPER, y a ambos los controla la última instrucción INK/PAPER que haya recibido.

```
 10 CIRCLE INK 2,128,88,50
 20 PLOT 0,0: DRAW INK 5,255,17
5
```
37

En este programa, debemos observar cómo todas las posiciones de carácter donde van cayendo los sucesivos cuadritos encendidos de la circunferencia, van tomando color rojo. En los puntos donde se cortan la circunferencia y la recta, toda la posición de carácter afectada toma el color azul para INK (cuadritos encendidos de la malla).

Para afianzar la idea de la alta resolución vamos a desarrollar y comentar unos programas:

```
  10 INPUT "color del borde? ";b
  20 INPUT "color del papel? ";p
  30 INPUT "color de la tinta? "
;t
  40 INPUT "abscisa inicial?  ";
x
  50 INPUT "ordenada inicial?  "
;y
  55 BORDER b: INK t: PAPER p: C
LS : PAPER p
  60 PLOT OVER 1,x,y
  61 LET d$=INKEY$
  62 IF d$="d" THEN GO SUB 500
  70 GO SUB 400
  75 IF y<10 THEN PRINT AT 0,5;"
"
  77 IF y<100 THEN PRINT AT 0,6;
" "
  80 PRINT AT 0,0;x;TAB 3;",";TA
B 4;y
 200 FOR f=0 TO 1                    38
 210 PLOT OVER f;x,y
 220 NEXT f
 230 GO TO 60
 300 IF x<0 THEN LET x=255
 310 IF x>255 THEN LET x=0
 320 IF y<0 THEN LET y=175
 330 IF y>175 THEN LET y=0
 340 RETURN
 400 LET a$=INKEY$
 410 IF a$="5" THEN LET x=x-1
 420 IF a$="8" THEN LET x=x+1
 430 IF a$="6" THEN LET y=y-1
 440 IF a$="7" THEN LET y=y+1
 450 GO SUB 300
 460 RETURN
 500 INPUT "abscisa? ";x1
 510 INPUT "ordenada? ";y1
 520 DRAW x1,y1
 522 LET x=x+x1: LET y=y+y1
 530 RETURN
```

De la línea 200 a la 220 tenemos un bucle de parpadeo. de la 300 a la 330 tenemos las condiciones en los bordes.

Para conseguir una serie de "pisos" en la pantalla con "agujeros" que se desplacen por ellos, podemos estudiar el siguiente desarrollo:

```
  10 BORDER 5: INK 1: PAPER 6
  20 PLOT 0,50: DRAW 255,0
  30 PLOT 0,100: DRAW 255,0
  40 PLOT 0,150: DRAW 255,0
  50 PLOT 0,3: DRAW 255,0
  75 LET x1=0
  80 FOR x=0 TO 255
 150 PLOT OVER 1,x,100: IF x>15
THEN LET e=x-15: PLOT OVER 1,e,1
00
 180 PLOT OVER 1,x,50: IF x>15 T
HEN LET e=x-15: PLOT OVER 1,e,50
 230 PLOT OVER 1,x,3: IF x>15 TH
EN LET e=x-15: PLOT OVER 1,e,3
 240 IF x<50 THEN GO TO 270
 250 PLOT OVER 1,x1,3: IF x1>15
THEN LET w1=x1-15: PLOT OVER 1,w
1,3
 260 LET x1=x1+1: IF x1=255 THEN
 LET x1=0
 270 PLOT OVER 1,255-x,150: IF x
>15 THEN LET e=255-x-15: PLOT OV
ER 1,e,150
 290 PLOT OVER 1,x,150: IF x>15
THEN LET e=x-15: PLOT OVER 1,e,1
50
 320 NEXT x
 330 GO TO 80
```

39

Es conveniente acostumbrarse a tratar de interpretar los listados antes de recurrir a los comentarios. En cualquier caso el programa anterior nos define, en la primera línea, los colores con los que vamos a trabajar. A continuación posicionan los cuadritos de arranque y finales de un conjunto de rectas horizontales. Entre las líneas 80 y 320 se establece un bucle para conseguir, gracias a la instrucción OVER, la sensación de que los agujeros se desplazan sobre las líneas, obteniéndose pantallas de este tipo:

Con estos listados se puede desarrollar juegos muy atractivos.

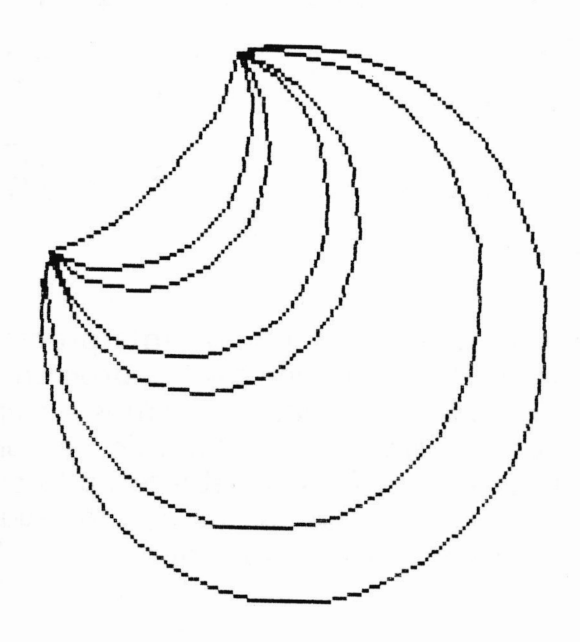
¿Se imagina a D. Pepe corriendo sobre estas líneas, tratando de saltar a través de los agujeros superiores y evitando caerse por los inferiores?

Para finalizar este epígrafe, diremos que el comando DRAW puede ser usado para trazar arcos de circunferencias siempre que éstos se midan en radianes.

Por ejemplo:

```
   5 INPUT "Pulse un valor de 0
 a 5 ";m
  10 PLOT 100,100
  20 DRAW 50,50,m
  30 GO TO 5
```

40



Antes de poder percibir el significado de sentencias tales como PEEK y POKE, necesitaremos conocer algunos conceptos generales sobre los procesos internos en un computador.

Todo se basa en la electrónica digital. No se preocupe. Ni nos hace falta, ni tocaremos esta rama de la técnica, pero sí necesitamos conocer aquellos aspectos que afectan directamente a un programador.

Digital: Perteneciente o relativo a los dedos. Esta es la definición que nos da el diccionario.

¿Y qué tienen que ver los dedos con los ordenadores? pensará Vd. Pues bien, todo viene de antiguo.

Los seres humanos tenemos diez dedos, cinco en cada mano, lo cual ha hecho normal contar cualquier cantidad como el número de veces que contiene, esa cantidad, los diez dedos. Así 24 naranjas corresponden a 2 veces 10 dedos más cuatro dedos: 2 x 10 + 4 = 24.

24 es un número, y el 2 y el 4 también lo son, pero los números comprendidos entre el 0 y 9 —ambos inclusive— son conocidos como "digitos" para darnos a entender que son aquéllos susceptibles de ser contados directamente por los dedos de nuestras dos manos. Es decir son los símbolos fundamentales del sistema decimal que es, lógicamente, el de uso corriente entre las personas.

En otro orden de cosas, pero profundamente ligado a lo anteriormente expuesto, los ordenadores solo pueden reconocer si un impulso eléctrico puede —o no— pasar por un interruptor. Si pasa le asignamos un 1, en caso contrario un 0. Vamos, que un ordenador tiene dos dedos y, consiguientemente, sólo dos dígitos básicos —1 y 0— de un sistema de numeración de base dos o binario. Esto quiere decir que, al tener dos dígitos, sólo podrá interpretar números formados por esos dígitos o, lo que es lo mismo, series de ceros y unos.

Cualquier número en base diez tiene su equivalente en base dos y, lógicamente, al revés. Por ejemplo, 17 en base diez se representa por 10001 en base dos. Puede estudiar esto con más profundidad. Vea la página 217 del manual.

Conocido ya porqué los ordenadores utilizan el sistema de numeración binario (o base dos) vamos a ver cómo lo manejan; para ello necesitamos conocer el significado de algunas palabras.

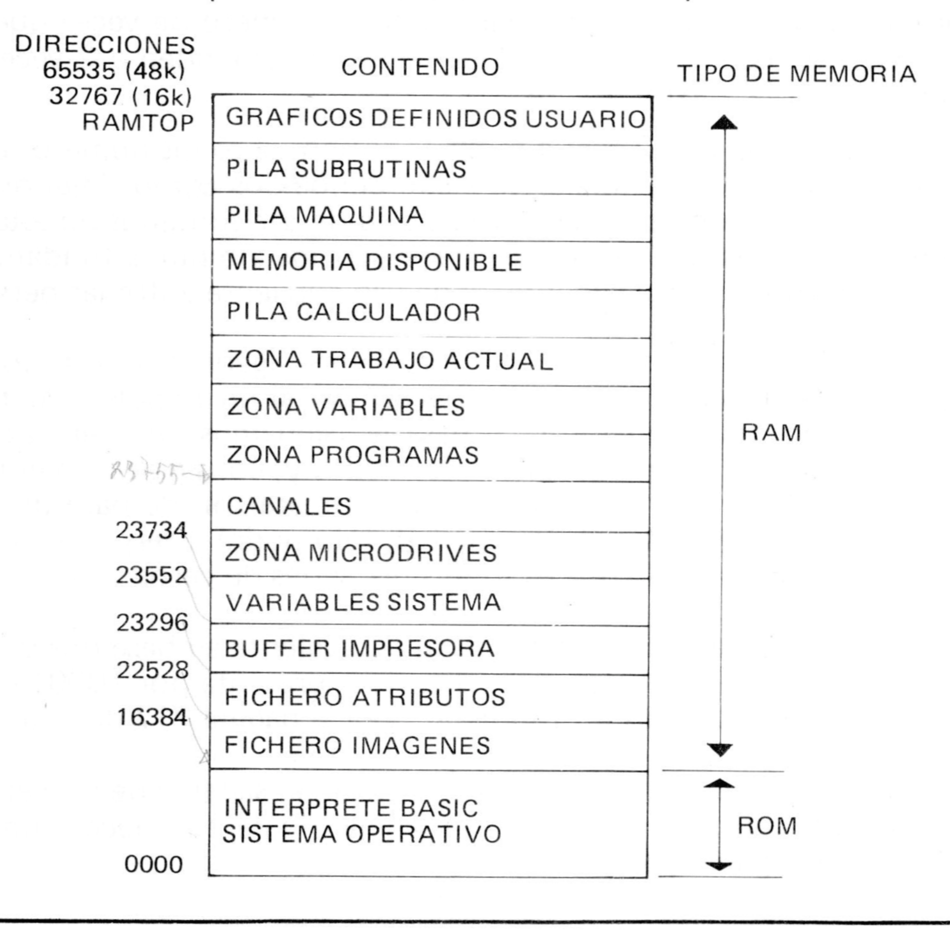
Un **bit** (se pronunca bit) nos indica uno de los dos estados —o dígitos— que un computador puede interpretar, y el número de bits que maneja dependerá de su microprocesador —verdadero cerebro del sistema— el cual, en el caso del Spectrum, es un Z80 de 8 bits que, consiguientemente, permite representar números comprendidos entre 0 y 255 —en base decimal— o, lo que es igual, puede hacer combinaciones de unos y ceros situados entre 00000000 y 11111111. A cualquiera de cada una de estas posibles combinaciones de 8 bits, se la conoce por un **byte** (se pronuncia bait) en su original vocablo anglosajón. O como un **octeto.**

Cada uno de estos bytes lo guarda el ordenador dentro de su memoria, en una posición definida, por lo que se conoce como una **dirección de memoria.**

La memoria está organizada como una secuencia lineal de direcciones o, sirva el ejemplo, como una calle con todas las casas en la misma acera, 8 pisos por casa y un solo vecino por piso. Estos vecinos solo pueden ser Ø ó 1.

Los números binarios ubicados en sus respectivas direcciones de memoria, son interpretados por el computador como números, caracteres o instrucciones dependiendo de su situación en la memoria.

La memoria está organizada en áreas, las cuales ocupan espacios fijos o variables dentro de un orden preestablecido denominado mapa de memoria:



Como se puede ver, hay dos zonas claramente definidas, la inferior corresponde a la memoria ROM (Read-Only Memory) y a la cual sólo se puede acceder para "leer" el contenio de la misma, y la memoria RAM (Randon Acces Memory) que puede servir tanto para escribir como para leer información. En otras palabras, la memoria RAM puede ser usada para almacenar o reclamar información, mientras que la ROM sólo sirve para reclamar aquella información que el fabricante haya introducido en ella y contiene, en el caso del Spectrum y entre otras cosas, el "interpreter" —o traductor simultáneo— del lenguaje BASIC.

La capacidad de memoria de un ordenador está en función de ia cantidad de información que pueda almacenar pero, siendo la unidad elemental de memoria el byte, tendremos que la memoria se mide en bytes o, más usualmente, en kilo-bytes, Kb o K sin más. Una K hace referencia a mil bytes, pero es simplemente una aproximación nemotécnica, ya que una K contiene exactamente 1024 bytes.

Así, un Spectrum de 16 K RAM indica que su capacidad de almacenamiento es de 16384 (16 x 1.024).

Dicho esto, podemos entender ya que cualquier posición de memoria puede guardar un número comprendido entre 0 y 255. Ahora veremos que un carácter se almacena como un conjunto de 8 bits.

En la página 183 del manual del Spectrum está el juego de caracteres. La primera columna es la denominada "código" (CODE) y está formada por numeros que van del 0 al 255 y la segunda es la denominada "carácter" (CHR). Esto significa que el código 65 está asimilado al carácter 65, de tal forma que si tecleamos PRINT CHR$ (65) obtendremos la letra A por respuesta. En definitiva cualquier posición de memoria puede contener un número entre 0 y 255, el cual será interpretado de acuerdo con la segunda columna de la tabla de caracteres del Spectrum.

Estas instrucciones están dirigidas a darnos información o manipular sobre las posiciones de memoria.

Pero, quizá, lo primero a saber es qué interpretación tendría —en español— estas dos palabras inglesas:

PEEK - Atisbar **POKE** - Hurgar

Bien, pues realmente esto es lo que nos permiten hacer. Con PEEK atisbamos lo que hay en una determinada posición de memoria definida por su dirección. Así, por ejemplo, si pretendemos conocer el contenido de la dirección de memoria 18121 bastaría con teclear PRINT PEEK 18121. Dado que PEEK siempre nos devolverá un número decimal —sin que esto implique que no pueda representar un número, un carácter o una instrucción— lo podríamos asimilar, si fuera de interés, a una variable numérica: LET a = PEEK 18121.

Quede claro que PEEK sólo "atisba" y, por tanto, no modifica ni altera nada de lo ya existente en memoria. Dicho con claridad: "no es peligrosa".

Con POKE la cosa cambia. Ya no se atisba, se hurga. Nos dirigimos a una dirección de memoria no para "atisbar" sino para "insertar" un determinado valor en esa posición y, por tanto es "peligrosa" ya que podríamos modificar posiciones situadas dentro de sectores de memoria vitales (memoria ROM).

Con POKE almacenamos un byte en **cualquier** posición de memoria, incluso en una que ya está habilitada por Vd., con lo cual perdería su contenido en favor del nuevo.

Como ejemplo vamos a teclear POKE 18121,65 almacenando, de esta forma, en la dirección 18121 el número decimal 65, con lo cual un PRINT CHR$ (PEEK 18121) nos da una A en pantalla.

Cabría pensar que un programador sólo necesita conocer la sintáxis del lenguaje de programación que utiliza y todas las palabras del mismo pero, es el caso que, a medida que domina las interioridades de su computador, puede obtener soluciones que serían inalcanzables sin tales conocimientos. Por tanto rogamos al lector un esfuerzo más, garantizándole que quedará sobradamente compensado.

Antes de ir más allá, recordaremos que el número más grande que se puede almacenar en una posición de memoria cualquiera está compuesto por 8 bits —que equivale a un byte— que corresponde a una combinación de ocho "ceros" y "unos". Por tanto, la combinación mínima vendrá dada por 00000000 y la más alta por 11111111, esta representación pertenece al sistema binario. Si lo expresáramos en el sistema decimal diríamos que en una posición de memoria se puede almacenar un número entre 0 y 255.

Así, si usamos una posición de memoria para contar, empezaremos por 0 y seguiremos sin dificultad hasta 255 pero, si aquí intentamos añadir otra unidad, no almacenaremos el número 256, sino que comenzaríamos otra vez a contar desde el cero. Pues bien, para alcanzar números superiores a 255, el computador habilita otra posición de memoria de tal forma que, cada vez que la anterior posición comienza a 0, la nueva aumenta una unidad. En estas condiciones, podríamos seguir contando en la primera posición de memoria como si nada hubiera pasado, mientras la segunda posición de memoria actúa como un contador del número de veces que la primera ha alcanzado 255.

La primera ubicación cuenta números entre 0 y 255 y es conocida como el **byte menos significativo** (LSB. Least Significant Byte) y la segunda ubicación cuenta de 256 en 256, y es conocida como el **byte más significativo** (MSB. Most Significant Byte).

Con un PEEK sobre el LSB, obtendremos un número entre 0 y 255 que representa justamente ese número, pero un PEEK sobre el MSB nos dará el número de veces que contiene 256. Esto, en términos que entienda el BASIC y siendo **n** la posición de memoria del LSB, sería:

PEEK n + 256 * PEEK (n + 1)

obteniendo de esta forma el número que está almacenado en estas dos posiciones de memoria, el cual, evidentemente, irá desde 0 hasta 65.535.

En sentido contrario podemos trocear un número para almacenarlo en dos ubicaciones de memoria; para ello lo dividiremos previamente entre 256 para saber el número de veces que lo contiene y almacenarlo en el byte más significativo (MSB) y entonces almacenar el resto en el byte menos significativo (LSB).

Esto es:

POKE n + 1, INT (N/256)
POKE n, N - 256 * INT (N/256)

Siendo **n** la dirección del LSB y **N** el número a trocear. Con POKE lo que hacemos es colocar los trozos adecuados en las direcciones de memoria seleccionadas.

Una vez en este punto echemos un vistazo a la página 174 del manual del Spectrum, segunda de las dedicadas a **las variables del Sistema**. En la primera línea se nos indica que las direcciones 23606 y 23607 están dedicadas a conocer las ubicaciones de memoria del juego de caracteres del Spectrum, de tal forma que, para obtener la dirección de memoria donde se inicia la tabla de caracteres, deberemos hacer la siguiente operación —de acuerdo con lo visto anteriormente—:

PEEK 23606 + 256 * PEEK 23607 + 256 = 15616

Los dos primeros sumandos representan un número decimal comprendido entre 0 y 65535 el cual, y según se indica en el manual, nos da la dirección de arranque (menos 256) del conjunto de caracteres.

El número de la dirección de arranque obtenido, sitúa el conjunto de caracteres dentro de la zona de memoria ROM que va desde la dirección 0 a la 16383, lo cual es lógico si tenemos en cuenta que el computador, una vez conectado, está en condiciones de trabajar gracias a su memoria ROM donde guarda toda la información básica del sistema y, entre ella, el juego de caracteres.

Este juego de caracteres, según el apéndice A del manual, va desde el **SPACE** (código 32) hasta el símbolo © (código 127).

Por otra parte, ya vimos que todo carácter está contenido en una malla de 8 x 8 cuadritos y que cada fila de esta malla es una combinación de "unos" y "ceros", para diferenciar los cuadritos encendidos de los apagados.

Por ejemplo:

0 0 0 0 0 0 0 0	1ª fila	0
0 0 1 1 1 1 0 0	2ª fila	60
0 1 0 0 0 0 1 0	3ª fila	66
0 1 0 0 0 0 1 0	4ª fila	66
0 1 1 1 1 1 1 0	5ª fila	126
0 1 0 0 0 0 1 0	6ª fila	66
0 1 0 0 0 0 1 0	7ª fila	66
0 0 0 0 0 0 0 0	8ª fila	0

En el ejemplo anterior hemos representado el modelo de malla de 8 x 8 de cuadritos encendidos y apagados que el ordenador tiene en su memoria ROM para la letra A y, dado que una dirección de memoria sólo puede guardar un byte, deducimos que cada carácter necesita 8 direcciones de memoria, una para cada fila, y de esta forma ser "memorizado".

La columna de la derecha son los números decimales que corresponde a los números binarios que representan los "ceros" y "unos" de cada fila de la malla de 8 x 8 y, en definitiva, serían los números decimales obtenidos al aplicar PEEK sobre cada una de las 8 direcciones de memoria en que está almacenado el carácter.

La cuestión ahora vendría al querer saber dónde está —o cuál es— la dirección de memoria de la primera fila de "ceros" y "unos" de cualquier carácter. Para responder a esto debemos recordar que la dirección de memoria donde se inicia la tabla de caracteres es: PEEK 23606 + 256 * PEEK 23607 + 256 = **da (dirección de arranque)** y también que la tabla comienza en el código 32 (SPACE). Por lo que de ambas cosas es fácil deducir que la dirección de memoria inicial de un carácter cualquiera —**c**— viene dada por:

dc = da + 8* (c-32)

Siendo **da** la dirección de arranque ya citada y **dc** y **c** la dirección y el código del carácter en cuestión.

Quiere decirse, que un **PEEK dc** nos daría el contenido, en decimal, de esa posición de memoria. Con una simple transformación a binario habríamos obtenido la combinación de "ceros" y "unos" que la memoria ROM guarda como modelo de la primera fila del carácter c.

En este momento ya conocemos **dónde** y **cómo** guarda en memoria el ordenador el conjunto de caracteres.

Una aplicación típica de todo lo tratado en este capítulo es el trazado de caracteres y cadenas de caracteres de gran tamaño.

Para llegar a un programa que nos haga tal trabajo, iremos por partes. Con la rutina que sigue, imprimiremos en pantalla la fila de arranque, en "ceros" y "unos" del carácter elegido como modelo para esta primera etapa.
(ver tabla de conversión en la página 81).

```
   10 INPUT "seleccione caracter
 ";c$
   20 IF LEN c$>1 THEN GO TO 10
   30 LET da=PEEK 23606+256*PEEK
 23607+256
   40 LET dc=da+8*(CODE c$-32)
   50 LET f=PEEK dc
   60 FOR i=0 TO 7
   70 LET b=f-2*INT (f/2): LET f=
 INT (f/2)
   80 PRINT AT 21,(20-2*i);b
   90 NEXT i
```

41

Para conseguir el carácter en cuestión completo, en su malla de "ceros" y "unos", solo tendríamos que añadir un bucle FOR/NEXT:

```
   10 INPUT "seleccione caracter
 ";c$
   20 IF LEN c$>1 THEN GO TO 10
   30 LET da=PEEK 23606+256*PEEK
 23607+256
   40 LET dc=da+8*(CODE c$-32)
   45 FOR j=0 TO 7
   50 LET f=PEEK (dc+j)
   60 FOR i=0 TO 7
   70 LET b=f-2*INT (f/2): LET f=
 INT (f/2)
   80 PRINT AT 21,(20-2*i);b
   90 NEXT i
  100 PRINT ''
  110 NEXT j
```

42

Al conocer este programa seleccione el carácter "s" y, si respondemos con una "y" al Scroll?, veremos el conjunto de 0 y 1 que forman la malla de 8 x 8 del carácter elegido, como se puede apreciar en la representación que sigue:

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0
0 1 0 0 0 0 0 0
0 0 1 1 1 0 0 0
0 0 0 0 0 1 0 0
0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0
```

Si allá donde aparezca un 0 dejamos su posición de carácter en color papel (PAPER) y donde aparezca un 1 colocamos una posición de carácter en el color (INK) que deseemos, habremos obtenido una representación a escala del carácter introducido en c$, de esta forma:

```
   5 PAPER 6: INK 1: BORDER 1
  10 INPUT "seleccione caracter
";c$
  20 IF LEN c$>1 THEN GO TO 10
  30 LET da=PEEK 23606+256*PEEK
23607+256
  40 LET dc=da+8*(CODE c$-32)
  45 FOR j=0 TO 7
  50 LET f=PEEK (dc+j)
  60 FOR i=0 TO 7
  70 LET b=f-2*INT (f/2): LET f=
INT (f/2)
  75 IF b=0 THEN GO TO 90
  80 PRINT AT 21,(20-2*i);"■"
  90 NEXT i
 100 PRINT ''
 110 NEXT j
```

43

Corriendo este programa obtendremos una impresión de este tipo:

```
 ■ ■ ■
■
 ■ ■ ■
        ■
■ ■ ■ ■
```

Pasar de la impresión de un solo carácter a una cadena de caracteres, se limita a añadir un bucle más para poder abarcar todos los caracteres de esa cadena:

```
  5 PAPER 6: INK 1: BORDER 1
 10 INPUT "entre texto  ";c$
 20 LET da=PEEK 23606+256*PEEK
23607+256
 30 FOR k=1 TO LEN c$
 40 LET dc=da+8*(CODE (c$(k TO
k))-32)
 45 FOR j=0 TO 7
 50 LET f=PEEK (dc+j)
 60 FOR i=0 TO 7
 70 LET b=f-2*INT (f/2): LET f=
INT (f/2)
 75 IF b=0 THEN GO TO 90
 80 PRINT AT 21,(20-2*i);"■"
 90 NEXT i
100 PRINT ''
110 NEXT j
120 NEXT k
```

44

En el mapa de memoria representado en el capítulo **"Procesos internos del Spectrum"** nos encontramos con el **"fichero de representación visual"** (Display file) que va desde la ubicación de memoria con dirección 16384 a la 22527, en cada una de estas posiciones de memoria se almacenará, desde que se conecta la máquina, una combinación de 8 "ceros" y "unos" —un byte— gracias a las cuales va a saber el ordenador qué cuadritos están apagados (0) o encendidos (1), en la zona controlada por INK y PAPER.

Como ya dijimos al principio, el ordenador considera dividida la pantalla del televisor en una malla de 256 cuadritos horizontales por 176 verticales, los cuales están apagados, inicialmente o, lo que es igual, todas las posiciones de memoria situadas entre 16384 y 22527 estarán ocupadas por 0.

Pruebe:

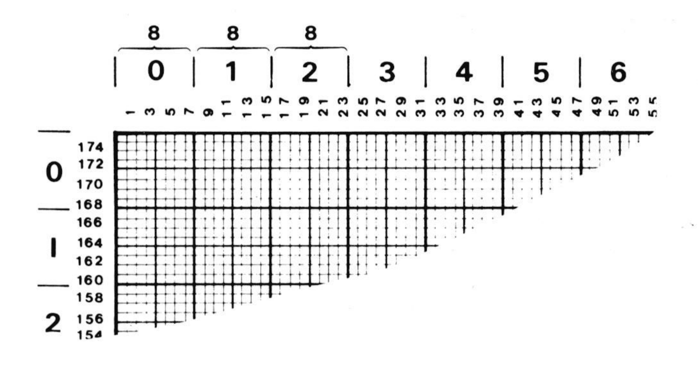
```
  10 INPUT "direccion a chequear
?  ";d
  15 CLS
  20 IF d<16384 OR d>22527 THEN
GO TO 10
  30 PRINT PEEK d
```

45

Ahora bien, a medida que se van produciendo impresiones en pantalla, el ordenador debe ir memorizando diferentes combinaciones de 0 y 1 en el "Display file". La cuestión, que podría ser de interés según el tipo de aplicación a desarrollar, sería **cómo** se produce el almacenamiento y **cómo encontrar** la dirección de memoria de un puntillo determinado de pantalla.

Empezamos por el **"cómo".**

A cada dirección de memoria del "Display file" corresponden 8 cuadritos de pantalla y sólo esos ocho cuadritos. El criterio de correspondencia es el siguiente:



La primera dirección almacenará la situación encendido/apagado (1,0) de los ocho primeros cuadritos de la primera fila de la malla de 256 x 176, los cuales obviamente corresponden a la primera fila de cuadritos de la malla de 8 x 8 de la primera posición de carácter. La siguiente dirección almacenará los ocho cuadritos siguientes de la misma fila y así sucesivamente hasta el final de la misma. En este momento tendremos 32 direcciones de memoria ocupadas que corresponden a las primeras 32 posiciones de carácter de la primera línea de caracteres.

A continuación se almacenan los ocho cuadritos primeros de la novena línea de cuadritos de la malla de 256 x 176 que corresponden a la primera fila de cuadritos de la malla de 8 x 8 de la primera posición de carácter de la segunda línea de caracteres. La siguiente dirección guardará los ocho cuadritos siguientes de la misma fila (la novena) y así sucesivamente hasta el final de la misma.

Después, y de la misma forma, los cuadritos de la fila 17 (tercera línea de caracteres) y así hasta llegar a la primera línea de cuadritos de la octava línea de caracteres.

En este punto se repite el proceso con todas las segundas filas hasta llegar a la octava línea de posiciones de carácter y así, hasta llegar a la octava fila de cuadritos de la octava línea de posiciones de carácter.

En este momento el ciclo se repite completo entre la primera fila de cuadritos de la novena línea de posiciones de carácter y la octava fila de cuadritos de la décimo sexta línea de posiciones de carácter.

Y el ciclo se repite entre la primera fila de cuadritos de la décimo séptima línea de posiciones de carácter y la octava fila de cuadritos de la vigésimo cuarta línea de posiciones de carácter (incluyendo, claro está, las dos líneas reservadas a entrada de datos y emisión de mensajes).

Para ver gráficamente todo lo anteriormente expuesto, introduzcamos en todas las direcciones de memoria del "Display file" una combinación de ocho "unos" (11111111); este número binario corresponden al 255 decimal, que será el código que usemos para hurgar (POKE) en las direcciones que nos interesen, una vez hecho esto procederemos a su impresión:

```
10 FOR d=16384 TO 22527
20 POKE d,255
30 NEXT d
40 FOR i=16384 TO 22527
50 PRINT AT 0,0;PEEK i
60 PAUSE 50
65 PRINT AT 0,0;"     "
67 PAUSE 20
70 NEXT i
```

Con estas instrucciones, modificamos de apagado (0) a encendido (1), todas las direcciones del "Display file". (líneas 10–30)

Con estas instrucciones obtendremos los números decimales, correspondientes a los binarios representativos de la situación en que se encuentran cada dirección del "Display file". (líneas 40–70)

46

Corra este programa.

Observe los dos ceros que aparecen en segundo y tercer lugar. Si Vd. ha comprendido el párrafo anterior, sabrá a qué son debidos.

Hasta aquí se ha visto **cómo** se produce la memorización del contenido de la pantalla. Ahora veremos como, dada una posición en pantalla de un grupo de 8 cuadritos determinados, localizar su dirección de memoria en el "Display file".

En la terminología aplicada en la explicación previa referente al "cómo", se han utilizado términos tales como **"Línea de posiciones de carácter"**. Para hacer referencia a la posición vertical de las mallas de 8 x 8 donde los caracteres se producen, y **"filas de cuadritos"** para especificar la situación relativa de los cuadritos dentro de las Líneas de Posición de carácter.

Utilizando estas expresiones diríamos que un grupo de 8 cuadritos está, por ejemplo, en la **línea de posición de carácter cuarta**, y en su tercera posición —de izquierda a derecha—, ocupando la séptima fila de cuadritos dentro de esa posición de carácter, de tal forma que si nosotros hacemos:

Línea de posición de carácter = **L** (varía entre 0 y 23)

Carácter de la línea = **C** (varía entre 0 y 31)

Fila de cuadritos = **F** (varía entre 0 y 7)

Cualquier grupo de 8 cuadritos dentro de la pantalla quedará posicionado con **L, C** y **F**.

Para llegar a una fórmula que nos dé la dirección de memoria de cualquiera de estos posibles grupos de 8 cuadritos, debemos partir de la primera dirección (16384) dedicada al control de pantalla. Después debemos averiguar el número de grupos completos de 8 líneas de carácter, que separan a nuestro grupo de 8 cuadritos del origen de la pantalla, cuyo valor puede ser uno de estos:

Caracteres en una línea	Líneas de posición de carácter		Filas carácter	
	0	x 8	= 0	
	1			
	2			
32 x	3			El grupo de 8 cuadritos está entre
	4			líneas 0 y 7
	5			
	6			
	7			
	8	x 8	= 2048	
	9			
	10			
32 x	11			El grupo de 8 cuadritos está entre líneas
	12			8 y 15
	13			
	14	x 8	= 4096	
	15			
	16			
	17			
	18			
32 x	19	x 8	= 6144	Entre líneas 16 y 23
	20			
	21			
	22			
	23			

La última posición posible es 6143

Esto, en términos matemáticos y BASIC, es igual a **2048* INT (L/8),** siendo **L** lo ya indicado. Una vez que sabemos en cuál de los tres sectores horizontales de la pantalla está situado el "byte" que nos interesa, deberemos determinar cuántas líneas completas (a 32 caracteres cada una) existen entre el comienzo del sector en cuestión y la posición de carácter donde se encuentra situado nuestro grupo de 8 cuadritos:

32 * (L - 8* INT (L/8)).

Ahora determinaremos el número de filas de cuadritos completas (a razón de 256 cuadritos cada una) que no han llegado a completar una línea de posición de carácter: 256*F. Por último, la fila de cuadritos incompleta que viene representada directamente por el valor de C.

Recopilando todos estos componentes llegamos a:

d = 16384 + 2048* INT (L/8) + 32* (L - 8 * INT (L/8))

Un modo de comprobar esta fórmula es dar valores a las variables **(L, C y F)** para unos valores conocidos de **d**. Este sería el caso, p. e., de la primera y última posiciones posibles de pantalla, las cuales tienen por direcciones de memoria d = 16384 y d = 22527.

```
  10 INPUT "linea de posicion de
 caracter? ";l
  20 INPUT "caracter en la linea
? ";c
  30 INPUT "fila de cuadritos? "
;f
  40 LET d=16384+2048*INT (l/8)+
32*(l-8*INT (l/8))+256*f+c
  50 PRINT "Direccion de memoria
 ";d
  60 GO TO 10
```

47

Pruebe estas dos series:

1°.– L = 0, C = 0, F = 0 . . . . d = 16384

2°.– L = 23, C = 31, F = 7 . . . . . d = 22527

Volviendo de nuevo al mapa de memoria podemos ver que, entre las direcciones 22528 y 23295, en encuentra situado el **fichero de atributos** (Attributes file) el cual almacena la información sobre el color (INK y PAPER), brillo (BRIGHT) y parpadeo (FLASH) de cada posición de carácter sobre un total de 24 x 32 posiciones de carácter.

Hay 768 direcciones de memoria disponibles en el "fichero de atributos" y un total de 768 posiciones de carácter en pantalla, lo cual implica que los atributos de cada posición de carácter son guardados y controlados por un solo byte o, como ya sabemos, por una combinación de ocho "ceros" y "unos".

Siguiendo el sistema aplicado con el "Display file", empezaremos por conocer **cómo** se produce el almacenamiento y a continuación analizaremos **cómo encontrar** la dirección de memoria de un carácter cualquiera. Más un tercer factor. De qué forma un byte controla los cuatro atributos: INK, PAPER, BRIGHT y FLASH.

El primer interrogante tiene una respuesta sencilla. El primer carácter de la primera línea de posición de carácter corresponde a la primera dirección del fichero de atributos, el segundo carácter de la misma fila a la segunda dirección de memoria y así hasta llegar al final —trigésimo segundo carácter— una vez en este punto, la siguiente dirección la ocupará, sencillamente, el primer carácter de la segunda línea de posiciones de carácter y, así, hasta el final de esta línea. El proceso se repite hasta la línea más baja de la pantalla.

Si mantenemos el significado de L, C y d dado anteriormente veremos que la dirección de memoria que ocupa el byte controlador de los atributos de un carácter cualquiera viene dado por:

$$d = 22528 + 32 * L + C$$

Fórmula evidente.

Respecto a la forma en que un solo byte controla los atributos, está en función de las posibilidades que ofrece cada atributo. Así, FLASH requiere **un bit** ya que sólo puede estar activado (0) o desactivado (1), BRIGHT **un bit** por la misma razón. INK y PAPER necesitan **tres bits** cada uno ya que son ocho los colores a controlar por cada comando. Por tanto, y de acuerdo con el manual (pág. 219), tendremos:

* Primer bit -------- indica situación de FLASH.
* Segundo bit ----- indica situación de BRIGHT.
* Tercer bit, Cuarto bit, Quinto bit ----- indican color de PAPER.
* Sexto bit, Séptimo bit, Octavo bit --- indican color de INK.

Con estos conocimientos podríamos deducir la situación de los atributos a través de un PEEK d, pero, gracias a la función ATTR, nos evitamos investigar la dirección de memoria —d— correspondiente a un carácter cualquiera situado en la línea L y en la posición C de esa línea. Para ello bastará aplicar **ATTR (L,C)** y obtendremos un número decimal comprendido entre 0 y 255, ya que corresponderá a una combinación binaria entre 00000000 y 11111111.

La rutina que sigue nos dará la situación de los atributos mediante la adecuada transformación del número decimal obtenido por ATTR (L, C):

```
  10 INPUT "linea del caracter?
";L
  20 INPUT "posicion del caracte
r en la linea? ";C
  30 LET a=ATTR (L,C)
  40 LET flash=INT (A/128): IF f
lash=0 THEN PRINT "FLASH DESACTI
VADO": GO TO 60
  50 PRINT "flash activado"
  60 LET bright=INT ((a-flash*12
8)/64): IF bright=0 THEN PRINT "
BRIGHT DESACTIVADO": GO TO 80
  70 PRINT "bright activado"
  80 LET paper=INT ((a-flash*128      48
-bright*64)/8)
  90 PRINT "paper ";paper
 100 LET ink=INT (a-flash*128-br
ight*64-paper*8)
 110 PRINT "INK ";ink
 120 GO TO 10
```

```
FLASH DESACTIVADO
BRIGHT DESACTIVADO
paper 7
INK 0
```

Para comprobar este programa una vez tecleado, introduzca PRINT INK 6; AT 10, 10; "■" y, después córralo con un GOTO 10/Enter.

Pulse STOP y ENTER. Pruebe ahora PRINT INK 3; BRIGHT 1; FLASH 1; PAPER 6; AT 10, 10; "■". Observe las diferencias entre ambos.

Como se puede observar en el mapa de memoria, a continuación del "fichero de atributos" viene el control del **"buffer" de la impresora y las variables del sistema**, que serán explicadas a continuación, y el resto de las zonas de memoria que no son objeto de este libro.

El "buffer" es como un depósito de donde sale su contenido sólo cuando llega al nivel adecuado. En este orden de cosas el **buffer de la impresora** almacena información hasta que ha completado una línea de 32 posiciones de carácter. Esto implica que necesita 32 x 8 bytes, que son en efecto, las direcciones disponibles en esta zona (de 23296 a 23551).

Las **variables del sistema** abarcan un conjunto de direcciones de memoria que van desde la 23552 a la 23734. Estas variables pueden necesitar uno o dos bytes. Como ya sabemos, un byte sólo puede contener números entre 0 y 255 y dos bytes pueden llegar de 0 a 65535.

Las variables del sistema, en función de sus respectivos valores mantienen la organización del sistema. Conocer estos valores o modificar su contenido puede ser muy útil.

El conjunto de estas variables se puede ver en la página 173 del manual del Spectrum. Conocer un poco más sobre algunas de ellas, servirá de ejemplo para profundizar en el estudio del resto.

Dirección 23560

Un PEEK sobre esta dirección nos da el código de la última tecla apretada.

Ejemplo:

Si Vd. teclea directamente PRINT PEEK 23560 seguido de ENTER, observará en el ángulo superior izquierdo cómo aparece el número 13, que es el código de la tecla ENTER (pág. 183 del manual), lo cual es lógico ya que esa tecla fué la última utilizada.

Con el fin de observar cómo una adecuada combinación de las posibilidades que ofrece la máquina, nos aumenta el campo de alternativas de uso del sistema, pruebe:

PRINT CODE INKEY$

seguido, claro está, de la tecla imperativa ENTER.

Igualmente obtendremos 13.

Pruebe el siguiente programa y convertirá su ordenador en una rudimentaria máquina de escribir:

```
 10 PAUSE 25: IF INKEY$="" THEN
GO TO 10
 20 PRINT CHR$ PEEK 23560;
 30 GO TO 10
```

49

Dirección 23561

Un PEEK sobre esta dirección nos da 35. Pruebe:

PRINT PEEK 23561

En la pantalla aparecerá 35, que es el tiempo, medido en 1/50 segundo, que debe estar apretada una tecla para iniciar su mecanismo de repetición automática.

Pruebe un POKE 23561, 0, seguido de ENTER por supuesto, y comprobará como la autorrepetición entra en funcionamiento mucho más tarde.

Un POKE 23561, 1, hace casi inmanejable el teclado.

Dirección 23562

PRINT PEEK 23562 nos dará 5, que es el tiempo, medido en 1/50 segundo, transcurrido entre dos repeticiones de una tecla en su proceso de autorrepetición.

Si prueba el siguiente programa obtendrá el tiempo, en cincuentavos de segundos, que mantiene apretada una tecla cualquiera:

```
   5 LET a=0
   6 CLS : LET b=a
   7 IF INKEY$="" THEN PRINT AT     50
 0,0;b: LET a=0: GO TO 7
  20 LET a=a+PEEK 23562
  30 GO TO 6
```

Si corre el programa y aprieta una tecla, verá aparecer en el ángulo superior izquierdo el número correspondiente al tiempo de la depresión.

El uso de esta dirección de memoria puede ser útil, entre otras cosas, para hacer reaccionar un gráfico cualquiera, con más o menos velocidad en función del tiempo que hemos mantenido apretada una determinada tecla.

Dirección 23606 (y 23607)

Estas direcciones y su utilidad fueron explicadas en el capítulo **Más sobre la memoria.**

Dirección 23609

Si tecleamos un PRINT PEEK 23609, ENTER, obtendremos Ø que es el tiempo que dura el ruido que simula la depresión de una tecla.

Si hacemos un POKE 23609, 10 queda un "click" muy audible.

Dirección 23625

Si en el transcurso de la ejecución de un programa cualquiera usted teclea:

PRINT PEEK 23625

Obtendrá el número de línea de programa donde está situado el cursor.

Dirección 23677

Un PRINT PEEK 23677 nos dará la abcisa del último cuadrito (pixel) posicionado en la pantalla.

Dirección 23678

Un PRINT PEEK 23678 nos dará la coordenada del último cuadrito (pixel) posicionado en la pantalla.

Un ejemplo aclaratorio de uso para las dos direcciones anteriores, podría ser:

```
10 PLOT 2,7
20 DRAW 100,75
30 PRINT PEEK 23677,PEEK 23678
```

51

Con lo cual obtendríamos una pantalla similar a ésta:

```
102                82
```

donde el número situado a la izquierda nos da el valor **x** del pixel donde acaba la recta y el número a la derecha nos dá el valor de **y**.

Si una vez que se ha introducido un programa, se teclea —o introduce a modo de instrucción— la siguiente sentencia:

PRINT ((PEEK 23627 + 256 * PEEK 23628) – (PEEK 23635 +256 * PEEK 23636))

Obtendremos el número de "bytes" que ocupa un programa.

En el caso de que Vd. desee ampliar la memoria disponible, a costa de perder la posibilidad de crear sus propios gráficos, teclee lo que sigue:

Para un Spectrum 48 K	**Para un Spectrum 16 k**
POKE 23675, 255	**POKE 23675, 255**
POKE 23676, 255	**POKE 23676, 255**
CLEAR 65534	**CLEAR 32766**

Dirección 23688

Si Vd. teclea esta instrucción:

```
PRINT AT 1Ø, 1Ø; "p";
```

y a continuación:

```
PRINT PEEK 23688
```

nos aparecerá el número 22, justo a continuación de la letra **p**, que corresponde a la diferencia existente entre 33 y la columna donde debe ser impreso —sobre la pantalla— el siguiente carácter.

Dirección 23689

Si repetimos: PRINT AT 10, 10; "p";

y ahora tecleamos:

```
PRINT PEEK 23689
```

obtendremos un 14 a la derecha de **p** y que representa la diferencia entre 24 y la línea donde debería ser impreso —sobre la pantalla— el siguiente carácter.

Estas dos últimas direcciones de memoria nos permiten averigurar dónde se producirá la siguiente impresión de pantalla.

Dirección 23692

Con un **POKE 23692, 255** desaparece el mensaje de **SCROLL?** y éste se produce automáticamente.

Pruebe estas líneas:

```
10 POKE 23692,255
20 FOR x=0 TO 100: PRINT "p"
30 NEXT x
```

52

Para finalizar, en ocasiones puede ser muy importante saber si un programa corre sobre un Spectrum 48K, para ello basta con utilizar:

PRINT PEEK 23732 + 256 * PEEK 23733

Si el programa corre sobre un Spectrum 48 K en el margen superior izquierdo de la pantalla aparecerá el número 65535.

TABLA DE CONVERSION DE DECIMAL-BINARIO

1 = 00000001	2 = 00000010	3 = 00000011	4 = 00000100	5 = 00000101
6 = 00000110	7 = 00000111	8 = 00001000	9 = 00001001	10 = 00001010
11 = 00001011	12 = 00001100	13 = 00001101	14 = 00001110	15 = 00001111
16 = 00010000	17 = 00010001	18 = 00010010	19 = 00010011	20 = 00010100
21 = 00010101	22 = 00010110	23 = 00010111	24 = 00011000	25 = 00011001
26 = 00011010	27 = 00011011	28 = 00011100	29 = 00011101	30 = 00011110
31 = 00011111	32 = 00100000	33 = 00100001	34 = 00100010	35 = 00100011
36 = 00100100	37 = 00100101	38 = 00100110	39 = 00100111	40 = 00101000
41 = 00101001	42 = 00101010	43 = 00101011	44 = 00101100	45 = 00101101
46 = 00101110	47 = 00101111	48 = 00110000	49 = 00110001	50 = 00110010
51 = 00110011	52 = 00110100	53 = 00110101	54 = 00110110	55 = 00110111
56 = 00111000	57 = 00111001	58 = 00111010	59 = 00111011	60 = 00111100
61 = 00111101	62 = 00111110	63 = 00111111	64 = 01000000	65 = 01000001
66 = 01000010	67 = 01000011	68 = 01000100	69 = 01000101	70 = 01000110
71 = 01000111	72 = 01001000	73 = 01001001	74 = 01001010	75 = 01001011
76 = 01001100	77 = 01001101	78 = 01001110	79 = 01001111	80 = 01010000
81 = 01010001	82 = 01010010	83 = 01010011	84 = 01010100	85 = 01010101
86 = 01010110	87 = 01010111	88 = 01011000	89 = 01011001	90 = 01011010
91 = 01011011	92 = 01011100	93 = 01011101	94 = 01011110	95 = 01011111
96 = 01100000	97 = 01100001	98 = 01100010	99 = 01100011	100 = 01100100
101 = 01100101	102 = 01100110	103 = 01100111	104 = 01101000	105 = 01101001
106 = 01101010	107 = 01101011	108 = 01101100	109 = 01101101	110 = 01101110
111 = 01101111	112 = 01110000	113 = 01110001	114 = 01110010	115 = 01110011
116 = 01110100	117 = 01110101	118 = 01110110	119 = 01110111	120 = 01111000
121 = 01111001	122 = 01111010	123 = 01111011	124 = 01111100	125 = 01111101
126 = 01111110	127 = 01111111	128 = 10000000	129 = 10000001	130 = 10000010
131 10000011	132 = 10000100	133 = 10000101	134 = 10000110	135 = 10000111
136 = 10001000	137 = 10001001	138 = 10001010	139 = 10001011	140 = 10001100
141 = 10001101	142 = 10001110	143 = 10001111	144 = 10010000	145 = 10010001
146 = 10010010	147 = 10010011	148 = 10010100	149 = 10010101	150 = 10010110
151 = 10010111	152 = 10011000	153 = 10011001	154 = 10011010	155 = 10011011
156 = 10011100	157 = 10011101	158 = 10011110	159 = 10011111	160 = 10100000
161 = 10100001	162 = 10100010	163 = 10100011	164 = 10100100	165 = 10100101
166 = 10100110	167 = 10100111	168 = 10101000	169 = 10101001	170 = 10101010
171 = 10101011	172 = 10101100	173 = 10101101	174 = 10101110	175 = 10101111
176 = 10110000	177 = 10110001	178 = 10110010	179 = 10110011	180 = 10110100
181 = 10110101	182 = 10110110	183 = 10110111	184 = 10111000	185 = 10111001
186 = 10111010	187 = 10111011	188 = 10111100	189 = 10111101	190 = 10111110
191 = 10111111	192 = 11000000	193 = 11000001	194 = 11000010	195 = 11000011
196 = 11000100	197 = 11000101	198 = 11000110	199 = 11000111	200 = 11001000
201 = 11001001	202 = 11001010	203 = 11001011	204 = 11001100	205 = 11001101
206 = 11001110	207 = 11001111	208 = 11010000	209 = 11010001	210 = 11010010
211 = 11010011	212 = 11010100	213 = 11010101	214 = 11010110	215 = 11010111
216 = 11011000	217 = 11011001	218 = 11011010	219 = 11011011	220 = 11011100
221 = 11011101	222 = 11011110	223 = 11011111	224 = 11100000	225 = 11100001
226 = 11100010	227 = 11100011	228 = 11100100	229 = 11100101	230 = 11100110
231 = 11100111	232 = 11101000	233 = 11101001	234 = 11101010	235 = 11101011
236 = 11101100	237 = 11101101	238 = 11101110	239 = 11101111	240 = 11110000
241 = 11110001	242 = 11110010	243 = 11110011	244 = 11110100	245 = 11110101
246 = 11110110	247 = 11110111	248 = 11111000	249 = 11111001	250 = 11111010
251 = 11111011	252 = 11111100	253 = 11111101	254 = 11111110	255 = 11111111

Probablemente usted habrá notado que la mayoría de los libros que tratan el tema de la programación, lo hacen desde el punto de vista de manejar "número" y "letras", olvidando la enorme capacidad del ordenador para transmitir "imágenes".

Evidentemente es necesario conocer la programación y las diferentes características de su computador pero, en igual o mayor medida, se deben manejar los efectos visuales para conseguir, p. e., presentaciones de programas que transmitan con eficiencia su contenido, de forma que resulten atractivos además de veraces.

Para conseguir tales efectos, debemos empezar por hacer los oportunos borradores de los efectos visuales que pretendemos conseguir y, para ello, hemos de disponer de facsímiles que nos reproduzcan la pantalla, tanto en alta como en baja resolución de gráficos.

Es evidente que Vd. se puede dibujar sus propias hojillas con las cuadrículas que le representen todas las posiciones de carácter y cuadritos (pixels) que tiene una pantalla del Spectrum pero, más probablemente, le compensará comprar las que, al efecto, hay en el mercado. Para desarrollar este epígrafe, nosotros trabajaremos en base al GRAFKIT.

El GRAFKIT está compuesto por un cuadernillo de 30 hojas que, simultáneamente, nos reproducen la pantalla del Spectrum en alta y baja resolución.

Tanto si trabaja a nivel de carácter (BR), o de cuadrito —pixel— (AR), los impresos permiten posicionarlos sin dificultad, ya que las cuadrículas van numeradas arriba, abajo, a derecha y a izquierda.

Debido a la "transparencia" de las hojas y a sus numeraciones —adecuadas al tipo de resolución— combinamos pixels y **posiciones de carácter** con claridad y orden.

Se adjuntan un par de impresos con el fin de poder seguir las explicaciones del epígrafe cómodamente. En caso de que le convenga adquirir un GRAFKIT completo (desarrollado por PARANINFO SOFT, S.A.), puede dirigirse a cualquier tienda especializada.

Veamos ahora un par de aplicaciones sobre el tratamiento de lo que llamaremos en lo sucesivo "información gráfica".

Gracias al GRAFKIT podemos posicionar con facilidad cualquier PRINT y/o PLOT.

Para ver la utilidad de esto, tratemos de crear una primera pantalla que nos sirva de introducción a un programa cualquiera desarrollado por nosotros. Para ello supongamos que nos interesa centrar en la pantalla el siguiente rótulo:

"KIT PARA GRAFICOS GRAFKIT. PROGRAMA DE PRESENTACION"

Vamos a tratar de cuadrar este título de acuerdo con nuestro gusto personal. Dentro de las muchas alternativas posibles, veamos un par de ellas:

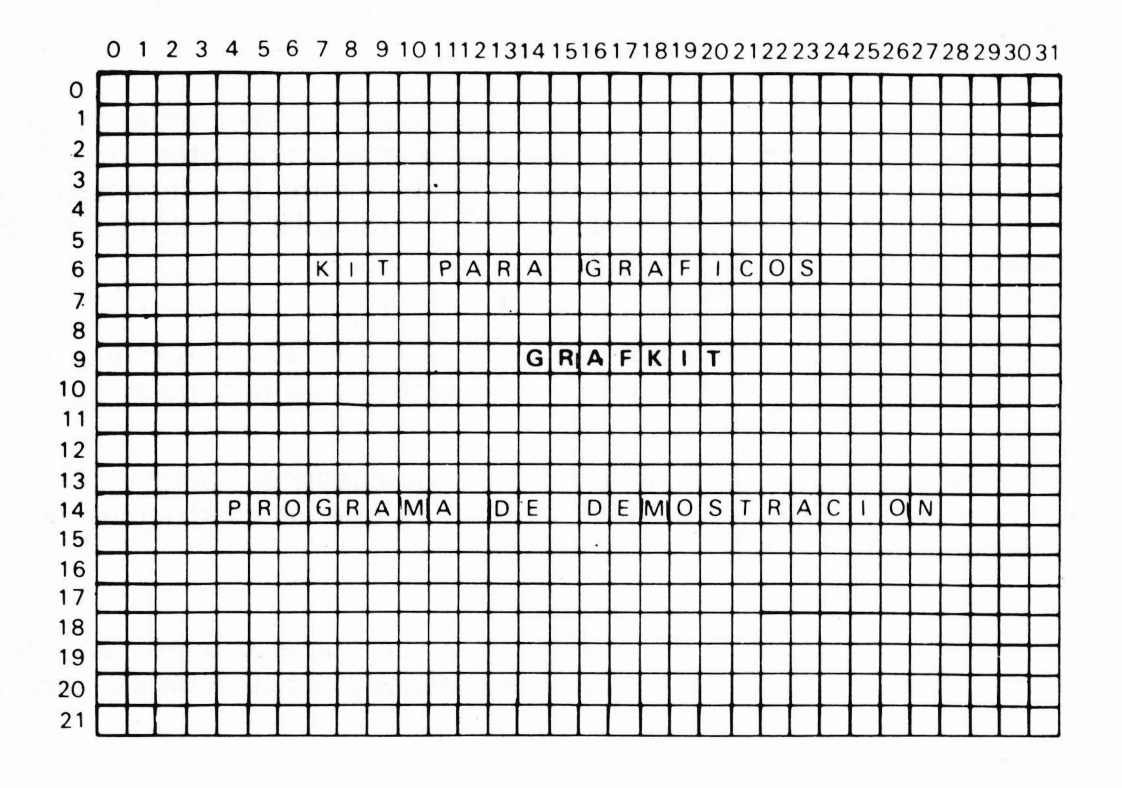
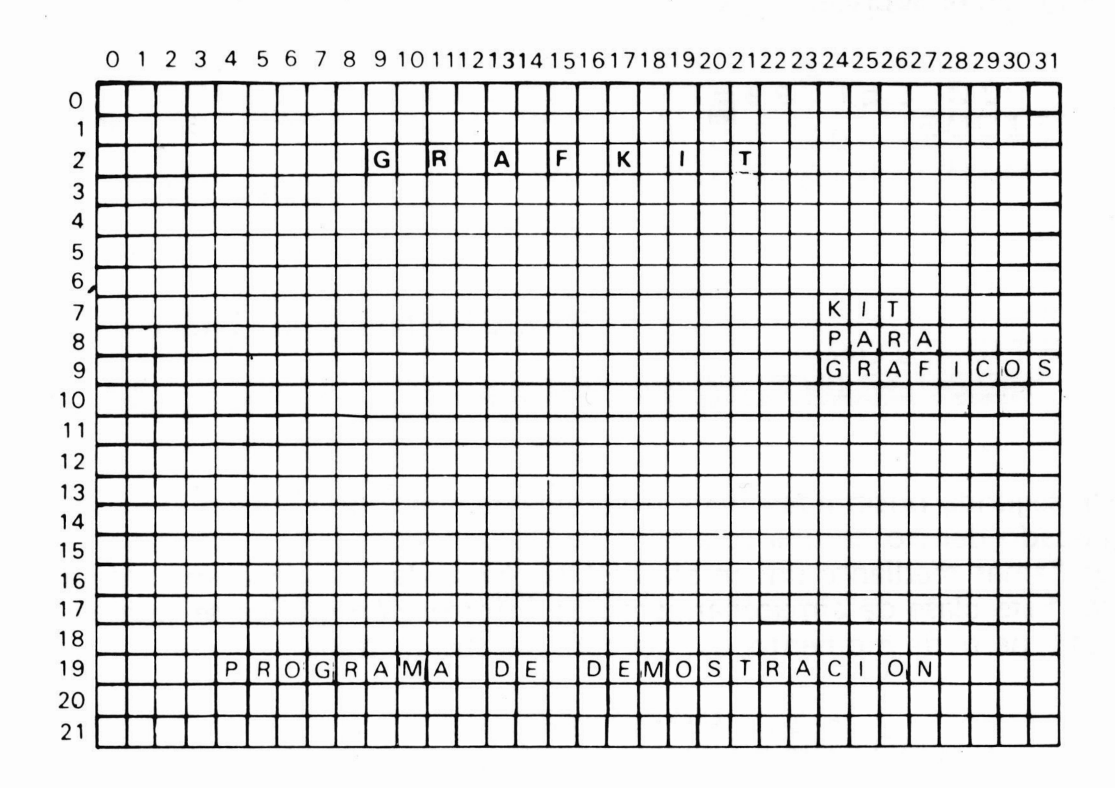


Figura A



Supongamos que preferimos la presentación de la figura A. Veamos en qué forma podemos programar esta pantalla. Un primer método puede consistir en usar sentencias PRINT AT:

```
  10 REM ***presentacion***
  20 PRINT AT 6,7;"KIT PARA GRAF
ICOS"
  30 PRINT AT 9,12;"GRAFKIT"
  40 PRINT AT 14,4;"PROGRAMA DE
DEMOSTRACION"
```

53

Otra forma consistiría en crear una cadena de caracteres, donde los espacios los contaríamos sobre el GRAFKIT:

```
  10 LET a$="            KIT PARA GRA
FICOS

                            GRAFKIT
                                              54



                        PROGRAMA DE DEMO
STRACION"
  20 PRINT AT 6,0;a$
```

Esta segunda posibilidad tiene la ventaja de que todo el texto, y su encuadre, ha quedado dentro de una sola variable de caracteres (a$ en este caso). Así podemos llamar mediante un simple PRINT AT toda la cadena en cualquier momento. Esta clase de almaçenamiento de información gráfica se revelará como muy útil, pero, de momento, así quedará nuestra pantalla:

```
            KIT PARA GRAFICOS

                 GRAFKIT



         PROGRAMA DE DEMOSTRACION
```

Para esta presentación hemos usado sólo la baja resolución de gráficos. Con el fin de combinar las sentencias PRINT y PLOT en el GRAFKIT y, simultáneamente, "adornar" esta pantalla podemos añadir estas líneas:

```
  10 LET a$="            KIT PARA GRA
FICOS

                            GRAFKIT

                        PROGRAMA DE DEM      55
OSTRACION"
  20 PRINT AT 6,0;a$
  50 LET a=29: LET b=27: LET z=2
5: LET d=23: LET e=21
  60 LET f=17
```

```
   70 FOR c=0 TO 112
   80 PLOT c,a: PLOT c,b: PLOT c,
z: PLOT c,d: PLOT c,e
   90 IF c<32 THEN PRINT AT f,c;"
█": PRINT AT 3,31-c;"█"
  100 PLOT 254-c,a: PLOT 254-c,b:
 PLOT 254-c,z: PLOT 254-c,d: PLO
T 254-c,e
  110 NEXT c
```

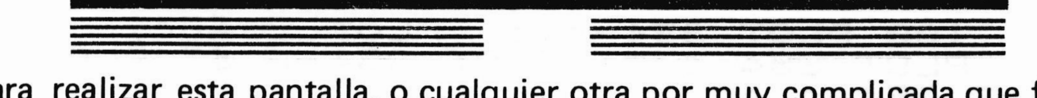
55

con lo cual, finalmente, tendremos esta impresión:

```
        KIT PARA GRAFICOS

             GRAFKIT


    PROGRAMA DE DEMOSTRACION
```



Para realizar esta pantalla, o cualquier otra por muy complicada que fuere, a través del GRAFKIT solo tendríamos que haberla dibujado previamente sobre el correspondiente impreso del GRAFKIT, según sea baja o alta resolución de gráficos y, si fuera necesario, superponer uno sobre otro de forma que combinemos con facilidad las sentencias PRINT y PLOT.

Para ver esto con más claridad, supongamos que queremos completar la pantalla de presentación añadiendo el siguiente dibujo en alta resolución:



En primer lugar dibujaríamos este perfil de cara sobre el GRAFKIT A. R. y a continuación la superpondremos sobre el GRAFKIT B. R. que contiene los textos de presentación y, tras introducir las oportunas modificaciones, tendríamos el siguiente listado:

```
  10 LET a$="           KIT PARA GRA
FICOS

                                GRAFKIT




                       PROGRAMA DE DEM
OSTRACION"
  20 PRINT AT 5,0;A$
  50 LET a=29: LET b=27: LET z=2
5: LET d=23: LET e=21
  60 LET f=17
  70 FOR c=0 TO 112
  80 PLOT c,a: PLOT c,b: PLOT c,
z: PLOT c,d: PLOT c,e
  90 IF c<32 THEN PRINT AT f,c;"
█": PRINT AT 3,31-c;"█"
 100 PLOT 254-c,a: PLOT 254-c,b:
 PLOT 254-c,z: PLOT 254-c,d: PLO
T 254-c,e
 110 NEXT c
 200 PLOT 111,127
 210 DRAW -4,0
 220 DRAW 0,-2
 230 DRAW -4,0
 240 DRAW -3,-3
 250 DRAW -3,0
 260 DRAW 0,-2
 270 DRAW -2,0
 280 DRAW 0,-2
 290 PLOT 94,117
 300 DRAW 0,-3
 310 DRAW 2,-3
 320 DRAW -3,-4
 330 DRAW 0,-12
 340 PLOT 92,95
 350 PLOT 92,94
 360 DRAW 3,-4
 370 DRAW -8,-13
```

56

```
380 DRAW 0,-2
390 DRAW 5,-4
400 DRAW 0,-2
410 PLOT 91,68
420 DRAW 3,-3
430 DRAW -3,0
440 DRAW 0,-2
450 PLOT 92,63
460 DRAW 2,-2
470 DRAW 0,-2
480 PLOT 93,-58
490 DRAW 0,-3
500 PLOT 94,-54
510 DRAW 3,-2
520 DRAW 8,-1
530 PLOT 105,-50
540 DRAW 2,-2
600 PLOT 112,128
610 DRAW 20,-1
620 DRAW 0,-2
630 DRAW 8,0
640 DRAW 2,-3
650 DRAW 11,-16
660 DRAW 3,-13
670 DRAW 0,-8
680 DRAW -6,-20
690 DRAW -3,0
700 DRAW -2,2
710 DRAW 1,-1
720 DRAW -2,-5
800 PRINT AT 10,16;"©"
```

56

con lo cual, finalmente, nos quedaría:



Esta breve introducción al manejo de la información gráfica, le ofrece una manera cómoda para diseñar sus propios gráficos, hacerlos aparecer en cualquier lugar de la pantalla y en el momento que lo desee o —más frecuentemente— utilizando la función RANDOMIZE. Todo lo explicado a lo largo de este libro, junto con el GRAFKIT, pone a su disposición la informática creativa, cuyo único límite viene impuesto por su voluntad y su imaginación.

Normalmente, cuando se escriben programas de aplicaciones profesionales, nos centramos exclusivamente en los datos —números y caracteres— de entrada y de salida, olvidándonos que los gráficos no sólo aumentan la claridad de interpretación, sino que además resaltan la profesionalidad del autor.

En este sentido, es usual ver "menús" de entrada de programas "serios" del tipo:

```
1 CALCULO
2 DIMENSIONADO
3 CURVAS
4 PESOS
5 MEMORIA




■ SELECCION
```

que corresponden a esta clase de listados:

```
  10 PRINT AT 8,8;"1 CALCULO";AT
 9,8;"2 DIMENSIONADO";AT 10,8;"3
 CURVAS";AT 11,8;"4 PESOS";AT 12
,8;"5 MEMORIA"
  50 PRINT FLASH 1;AT 21,8;"■ SE
LECCION"
  60 INPUT A
  70 GO TO A*1000
```

57

A veces la razón de estas pantallas tan "asépticas" suele ser la imposibilidad de consumir ni un solo byte más. No obstante, con un par de instrucciones, podemos modificar la pantalla ostensiblemente:

```
SELECCIONE ACTIVIDAD

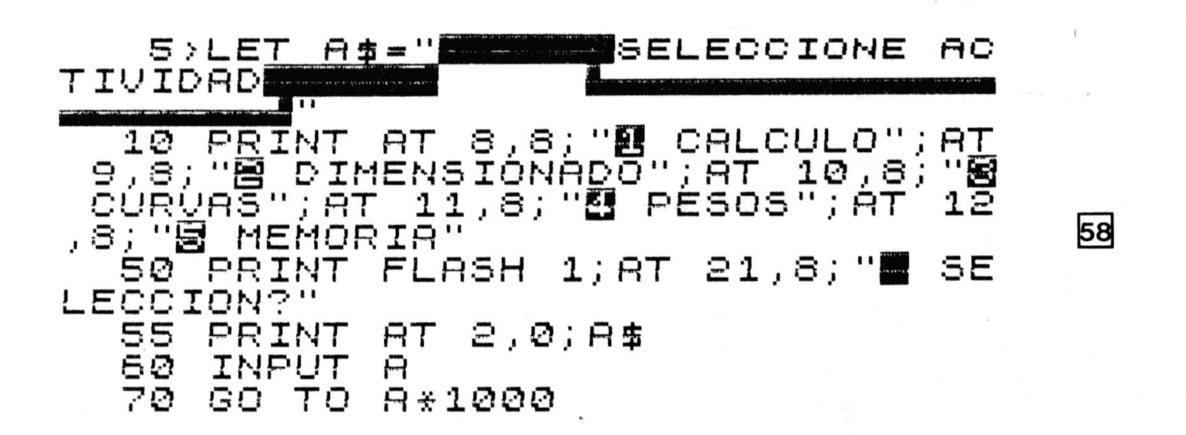


        1 CALCULO
        2 DIMENSIONADO
        3 CURVAS
        4 PESOS
        5 MEMORIA




        ■ SELECCION?
```

y para conseguir esta presentación el programa anterior cambiaría a:



```
   5>LET A$="      SELECCIONE AC
TIVIDAD      "
  10 PRINT AT 8,8;"1 CALCULO";AT
 9,8;"2 DIMENSIONADO";AT 10,8;"3
 CURVAS";AT 11,8;"4 PESOS";AT 12
,8;"5 MEMORIA"
  50 PRINT FLASH 1;AT 21,8;"■ SE
LECCION?"
  55 PRINT AT 2,0;A$
  60 INPUT A
  70 GO TO A*1000
```

58

Otra aplicación del GRAFKIT, viene dada por las mallas de 8 x 8 pixels representativas de un carácter y sobre las cuales el usuario puede definir sus propios gráficos. Sobre una de estas retículas, y según lo estudiado en el capítulo "Creando nuestros propios caracteres gráficos", vamos a crear este extraño carácter sobre la letra A:

BIN	0	0	0	0	0	1	1	1
BIN	0	0	0	1	1	1	1	1
BIN	0	0	1	1	1	1	1	1
BIN	0	1	1	1	1	1	1	1
BIN	0	1	1	1	1	1	1	1
BIN	1	1	1	1	1	1	1	1
BIN	1	1	1	1	1	1	1	1
BIN	1	1	1	1	1	1	1	1

Ver tabla de conversión decimal-binario (pág. 81).

Estos números binarios corresponden a la siguiente serie decimal: 7, 31, 63, 127, 255, 255, 255, con lo cual podemos crear nuestro carácter de acuerdo con la siguiente rutina:

```
 250 POKE USR "A",7: POKE USR "A
"+1,31: POKE USR "A"+2,63: POKE
USR "A"+3,127: POKE USR "A"+4,12      59
7: POKE USR "A"+5,255: POKE USR
"A"+6,255: POKE USR "A"+7,255
```

Obviamente, es más cómodo utilizar números decimales que binarios para crear caracteres gráficos.

Una vez aclarado esto, y si Vd. tiene paciencia, podrá ver en su pantalla un bonito grafismo, tecleando el siguiente listado:

```
 250 POKE USR "A",7: POKE USR "A
"+1,31: POKE USR "A"+2,63: POKE
USR "A"+3,127: POKE USR "A"+4,12
7: POKE USR "A"+5,255: POKE USR
"A"+6,255: POKE USR "A"+7,255         60
 260 POKE USR "B",224: POKE USR
"B"+1,248: POKE USR "B"+2,252: P
OKE USR "B"+3,254: POKE USR "B"+
4,254: POKE USR "B"+5,255: POKE
USR "B"+6,255: POKE USR "B"+7,25
5
 265 POKE USR "C",255: POKE USR
"C"+1,255: POKE USR "C"+2,255: P
OKE USR "C"+3,127: POKE USR "C"+
4,127: POKE USR "C"+5,63: POKE U
SR "C"+6,31: POKE USR "C"+7,7
```

```
 270 POKE USR "D",255: POKE USR
"D"+1,255: POKE USR "D"+2,255: P
OKE USR "D"+3,254: POKE USR "D"+
4,254: POKE USR "D"+5,252: POKE
USR "D"+6,248: POKE USR "D"+7,22
4
 280 POKE USR "E",1: POKE USR "E
"+1,2: POKE USR "E"+2,4: POKE US
R "E"+3,8: POKE USR "E"+4,16: PO
KE USR "E"+5,32: POKE USR "E"+6,
64: POKE USR "E"+7,128
 290 POKE USR "F",128: POKE USR
"F"+1,64: POKE USR "F"+2,32: POK
E USR "F"+3,16: POKE USR "F"+4,8
: POKE USR "F"+5,4: POKE USR "F"
+6,2: POKE USR "F"+7,1
 300 POKE USR "G",128: POKE USR
"G"+1,192: POKE USR "G"+2,224: P
OKE USR "G"+3,240: POKE USR "G"+
4,248: POKE USR "G"+5,252: POKE
USR "G"+6,254: POKE USR "G"+7,25
5
 310 POKE USR "H",1: POKE USR "H
"+1,3: POKE USR "H"+2,7: POKE US
R "H"+3,15: POKE USR "H"+4,31: P       60
OKE USR "H"+5,63: POKE USR "H"+6
,127: POKE USR "H"+7,255
 320 FOR q=0 TO 3: POKE USR "I"+
q,255: NEXT q: POKE USR "I"+4,24
8: POKE USR "I"+5,252: POKE USR
"I"+6,254: POKE USR "I"+7,255
 330 FOR q=0 TO 3: POKE USR "J"+
q,255: NEXT q: POKE USR "J"+4,31
: POKE USR "J"+5,63: POKE USR "J
"+6,127: POKE USR "J"+7,255
 340 FOR q=0 TO 3: POKE USR "K"+
q,255: NEXT q: POKE USR "K"+4,3:
 POKE USR "K"+5,3: POKE USR "K"+
6,1: POKE USR "K"+7,1
 350 FOR q=0 TO 3: POKE USR "L"+
q,255: NEXT q: POKE USR "L"+4,19
2: POKE USR "L"+5,192: POKE USR
"L"+6,128: POKE USR "L"+7,128
 360 POKE USR "M",255: POKE USR
"M"+1,255: FOR q=2 TO 5: POKE US
R "M"+q,127: NEXT q: POKE USR "M
"+6,63: POKE USR "M"+7,63
 370 POKE USR "N",255: POKE USR
"N"+1,255: FOR q=2 TO 5: POKE US
R "N"+q,254: NEXT q: POKE USR "N
"+6,252: POKE USR "N"+7,252
```

```
 380 FOR q=0 TO 2: POKE USR "O"+
q,0: NEXT q: POKE USR "O"+3,63:
POKE USR "O"+4,127: POKE USR "O"
+5,255: POKE USR "O"+6,127: POKE
 USR "O"+7,63
 390 FOR q=0 TO 2: POKE USR "P"+
q,0: NEXT q: POKE USR "P"+3,252:
 POKE USR "P"+4,254: POKE USR "P
"+5,255: POKE USR "P"+6,254: POK
E USR "P"+7,252
 400 FOR q=0 TO 2: POKE USR "Q"+
q,0: NEXT q: FOR q=3 TO 7: POKE
USR "Q"+q,255: NEXT q
 410 POKE USR "R",63: POKE USR "
R"+1,0: POKE USR "R"+2,63: FOR q
=3 TO 7: POKE USR "R"+q,255: NEX
T q
 420 POKE USR "S",252: POKE USR
"S"+1,0: POKE USR "S"+2,252: FOR
 q=3 TO 7: POKE USR "S"+q,255: N
EXT q
 430 POKE USR "T",255: FOR q=1 T      60
O 7: POKE USR "T"+q,0: NEXT q
 500 FOR y=2 TO 22 STEP 10
 510 FOR x=1 TO 15 STEP 7
 520 GO SUB 1000
 530 NEXT x
 540 NEXT y
 550 STOP
1000 PRINT PAPER 7; INK 5;AT x,y
;"A";AT x,y+7;"B"
1010 PRINT PAPER 7; INK 5;AT x,y
+3;"TT"
1020 PRINT PAPER 5; INK 0;AT x,y
+1;"F";AT x,y+6;"E"
1030 PRINT PAPER 5; INK 7;AT x,y
+2;"H";AT x,y+5;"G"
1040 PRINT PAPER 5;AT x+1,y;" ";
AT x+1,y+7;" "
1050 PRINT PAPER 5; INK 1;AT x+1
,y+1;"H";AT x+1,y+6;"G";AT x+2,y
;"H";AT x+2,y+7;"G"
1060 PRINT INK 1; PAPER 7;AT x+1
,y+2;"G";AT x+1,y+5;"H";AT x+2,y
+1;"IK";AT x+2,y+5;"LJ";AT x+3,y
;"C█GMNH█D";AT x+4,y;"OQQRSQQP"
1070 PRINT PAPER 7; INK 2;AT x+1
,y+3;"AB": PRINT PAPER 1; INK 2;
AT x+2,y+3;"CD"
1080 RETURN
```

Estúdielo con atención y comprobará que el campo de aplicación de un computador sobrepasa cualquier previsión que usted haya hecho y, lo más importante, nada de lo que Vd. se proponga hacer con estas máquinas queda fuera de su alcance.

Al llegar a este punto, y releyendo estas líneas desde el principio, pienso que tal vez he cubierto el objetivo que me propuse en la primera palabra de mi primer libro. Este objetivo no era ambicioso y, desde luego, no pretendía confeccionar un tratado enciclopédico.

Muy al contrario, mi intención era simple: escribir de una forma sencilla y clara, sobre temas informáticos, para personas que no hubieran tenido contacto alguno con el mundo de los ordenadores.

Para ello me he apoyado en un ordenador económico, eficaz y al alcance de cualquier bolsillo.

Si al concluir la lectura de este libro —y tal vez, de los anteriores— usted ha conseguido establecer su propio rumbo dentro del conjunto de posibilidades que ofrece la Informática, si he conseguido que le pierda el "respeto" al computador, me doy por satisfecho.

Por último, antes de despedirme de Vd. querido lector, quisiera transmitirle mi convencimiento de que, aunque "otros" hayan hecho mucho, Vd. lo puede hacer mejor.

Otros libros del mismo autor publicados por



ZX81 CURSO DE PROGRAMACION BASIC

TERCERA edición. 128 páginas. 21 x 27 cm. Rústica plastificada
ISBN: 84-283-1277-X

Presenta el BASIC como el más popular de los lenguajes de programación, por sus propias características y porque puede ser "entendido" por la casi totalidad de los microordenadores disponibles en el mercado.
Dirigido especialmente a quiénes desean aprender a manejar un ordenador y carecen de conocimientos de programación y de inglés.
El ordenador ZX81 de Sinclair opera en lenguaje BASIC, tiene un precio asequible y dispone de gran cantidad de programas para él. Por ello ha sido elegido por el autor para el "Curso de Programación".

COMO PROGRAMAR SU SPECTRUM Y TIMEX 2068

QUINTA edición. 132 páginas. 1 esquema de conexiones del Spectrum. 21,5 x 28 cm. Rústica plastificada
ISBN: 84-283-1305-9

Trata de popularizar la programación por cuanto implica una aproximación a la forma de razonamiento y utilización de las máquinas. Los temas se estructuran en base a:

La forma de utilizar el Spectrum — Cómo programarlo — Prácticas que pueden realizarse en él

De interés para programadores y principiantes a la vez y, en general para quienes tengan como herramienta base el microcomputador Sinclair o Spectrum.

Otros libros sobre "LENGUAJE BASIC" publicados por:



BASIC. CURSO ACELERADO

Por Claude J. de Rossi
TERCERA edición. 224 páginas. 35 ilustraciones. 15,5 x 21,5 cm. Rústica plastificada.
ISBN: 84-283-1289-3

Para quienes tienen urgencia por aprender a programar en BASIC. Parte de conceptos fundamentales hasta llegar a los más complicados y sofisticados. Contiene numerosos ejercicios y más de 350 ejemplos prácticos.
Su lectura detenida permite al lector alcanzar la habilidad necesaria para practicar en cualquier ordenador con lenguaje BASIC, aun careciendo de conocimientos previos de programación.

BASIC. INTRODUCCION A LA PROGRAMACION

Por Jean-Claude Larreche
TERCERA edición. 132 páginas. 5 ilustraciones. 15,5 x 21,5 cm. Rústica plastificada.
ISBN: 84-283-1215-X

Dirigido a profesionales, estudiantes y futuros especialistas en Informática. Consta de cuatro partes esenciales:

- Introducción a la programación
- Definición del lenguaje en BASIC
- Realización de un programa en BASIC
- Programas útiles en BASIC, para resolver en ordenador

Todo ello con numerosos ejercicios y ejemplos prácticos que permiten al lector aprender a escribir sus propios programas y dar instrucciones correctas al ordenador.

BASIC. PROGRAMACION DE MICROORDENADORES

Por Alain Checroun
TERCERA edición. 112 páginas. 9 ilustraciones. 15,5 x 21,5 cm. Rústica plastificada
ISBN: 84-283-1244-3

Define los elementos que constituyen el sistema de información en torno a un microordenador y destaca la importancia del lenguaje de programación en todos sus niveles.
Especialmente estudia el BASIC por ser uno de los lenguajes más extendidos. Un libro para quienes conocen poco o nada sobre ordenadores, con abundante bibliografía, que introduce la noción de fichero y su tratamiento con BASIC.

BASIC PARA NIÑOS
Con notas didácticas para padres y educadores

Por Sofía Watt y Miguel Mangada
SEGUNDA edición. 128 páginas. 86 ilustraciones. 15,5 x 21,5 cm. Rústica plastificada.
ISBN: 84-283-1327-X

Un material sencillo en una *obra didáctica* en la que los dibujos juegan un importante pape1. Permite al niño acercarse al fascinante mundo del ordenador y contiene numerosos ejercicios y ejemplos prácticos, relacionados con su entorno infantil, que abren al niño la puerta de lo que en el futuro será su herramienta de trabajo.
El adulto juega el importante papel de "guía" y estímulo y no de profesor. Los capítulos se estructuran en progresión ascendente de dificultad conceptual, lo que permite el aprendizaje.

COMO USAR LOS COLORES Y LOS GRAFICOS EN EL SPECTRUM

Este libro está escrito como una introducción al uso de los gráficos y el color en los computadores populares y ha sido desarrollado en base a un ZX Spectrum. Su objetivo es ayudar al lector a transformar sus ideas en programas llenos de color y movimiento.

TODOS LOS PROGRAMAS QUE SE DESARROLLAN EN ESTE LIBRO SE INCLUYEN EN UN CASETE QUE PUEDE ADQUIRIRSE OPCIONALMENTE. SOLICITELO A SU PROVEEDOR HABITUAL O A PARANINFO, S.A.



Magallanes, 25 - Madrid-15

ISBN: 84-283-1311-3